ioProgrammo

ANTEPRIMA: IIS 7.0
TUTTE LE PROMESSE DELLA NUOVA VERSIONE DEL SERVER WEB DI MICROSOFT

Rivista + "Le grandi guide di ioProgrammo" n°7 a € 12,90 in più

VERSIONE PLUS ☑ RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD ☐ RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • OTTOBRE 2006 • ANNO X, N.10 (107)

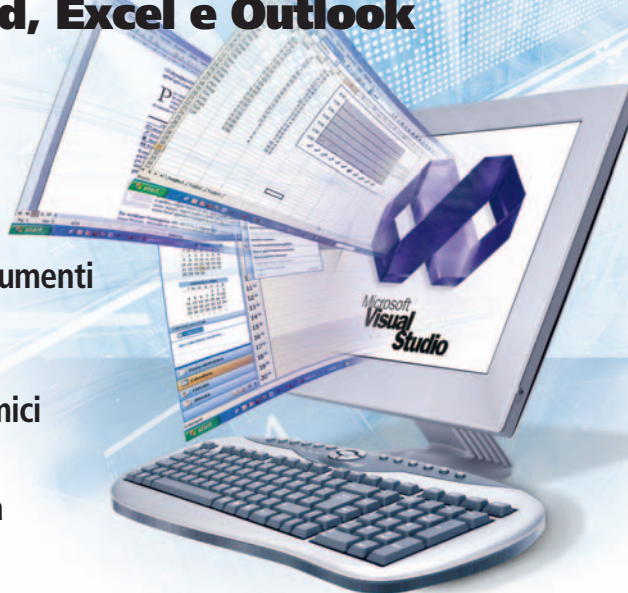
VISUAL STUDIO 2005 incontra MS Office

Sfrutta tutta la potenza di .NET ed aggiungi nuove funzioni a Word, Excel e Outlook

STRUMENTI
Cosa sono e come funzionano i Visual studio Tools for Office

TEORIA
Dai Bookmark ai NamedRange, ecco i controlli che "pilotano" i documenti

TECNICA
Progetta un modello da usare per la creazione di fogli elettronici dinamici

FUNZIONI AVANZATE
Sviluppa un "add-in" che si integra nell'interfaccia delle applicazioni

FATTI LA SEGRETARIA PER LIVE MESSENGER

Ecco come si crea un plugin per MSN. In più: l'esempio completo su come programmare un bot che risponde anche quando tu non ci sei

PHP SI RINNOVA

ARRIVA LO ZEND FRAMEWORK

Decine di classi "precotte" da utilizzare subito. Ti spieghiamo come usarle nei tuoi progetti

VISUAL BASIC
SISTEMA OPERATIVO CONTROLLALO COSÌ
Con le API e il SubClassing, Windows lo riprogrammiamo noi...

VISUAL BASIC.NET
CREA DA SOLO IL TUO STARTER KIT
Come sviluppare le soluzioni "pilota" che ti semplificano la vita

SICUREZZA Alla scoperta della crittografia quantistica. Lo stato dell'arte nel campo della tutela della privacy

SPECIALE RUBY ON RAILS
COSTRUISCI IN 10 MINUTI UNA MESSAGEBOARD
PREMIATO COME IL MIGLIOR FRAMEWORK PER LO SVILUPPO WEB NEL 2005

ASP.NET
DATI SEMPRE REALI NELLE FORM
Evita l'inserimento di informazioni fasulle usando le tecniche giuste!

AREE PROTETTE NEL TUO SITO
Usa il .NET framework per gestire in modo sicuro login e permessi

NOVITÀ
CHI C'È DIETRO A QUEL BROWSER?
WCF, ecco come Microsoft certificherà l'identità digitale

JAVA
DEMONI E SERVIZI PER TUTTI I GUSTI
Software residente in Windows o Unix puoi svilupparlo così…

L'ABITO CHE FA IL MONACO!
Quando la presentazione conta, usa Jasper Reports

SQL SERVER
LE RELAZIONI RIFLESSIVE
Che fare quando una tabella richiama se stessa?

PATTERN
LA CATENA DELLE RESPONSABILITÀ
Il modello perfetto per creare classi indipendenti fra loro

Anno X - N.ro 10 (107) - Ottobre 2006 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Redazione: Fabio Farnesi
Collaboratori: C. Bellucci, L. Buono, D. De Michelis, F. Grimaldi, M. Scala, A. Pelleriti, M. Locuratolo, L. Corias

Segreteria di Redazione: Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI

ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 20% sul prezzo di copertina di €75,90 · ioProgrammo con Libro (11 numeri) €75,90 sconto 30% sul prezzo di copertina di €108,90 Offerte valide fino al 30/11/06 Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ tel. **02 831212**
@ e-mail: **servizioabbonati@edmaster.it**

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Settembre 2006

▼ IL GRANDE RITORNO

Così a quanto pare ci siamo! Borland tornerà a sviluppare e distribuire i propri IDE e compilatori. La notizia è di quelle che fanno sorridere di piacere i pionieri della programmazione. Delphi, così come il Borland C++ hanno significato veramente tanto per la storia dello sviluppo. La loro stessa esistenza ha fornito alla concorrenza un motivo in più per progettare buoni prodotti. Perchè non si può certo dire che Visual Studio sia meno che un ottimo prodotto, ma gli mancava lo stimolo di un concorrente altrettanto agguerrito come lo è stato Borland per tanti anni. Non che cambi tantissimo dal punto di vista della tecnologia pura. I nuovi strumenti di Borland supportano abbondantemente il .NET Framework, ed è ovvio che sia così, vista la direzione che sta prendendo lo sviluppo del sistema operativo.
Perciò la battaglia non si svolgerà tanto sul piano della tecnologia quanto sul piano degli IDE e dei compilatori. Altrettanto interessante è la strategia che Borland ha scelto per tornare sul mercato. Sarà resa disponibile gratuitamente, infatti, una nuova versione denominata "Turbo" che, con la stessa logica delle release Express di Microsoft, rappresenterà un entry point per chi vuole provare i prodotti.
Non c'è mai stato tanto fermento nel campo dello sviluppo come in questi ultimi tempi. Da un lato sul fronte dei database è guerra aperta fra SQL Server, Oracle, Postgres e MySql, dall'altra parte l'arrivo dei prodotti di Borland apre una nuova linea di frontiera. In mezzo ci sta un mondo fatto di API, di prodotti intermedi, e di tool di analisi e progettazione e non si contano più i framework di supporto.
Persino Google si è messo a fare concorrenza a Sourceforge con un suo sito dedicato all'hosting per programmatori. Tutto questo non può che farci piacere. Aspettiamo che allo stesso fermento che movimenta il lato tecnologico dello sviluppo faccia riscontro un fermento nelle aziende. Aspettiamo che tornino ad investire in tecnologia, perché è anche in questo modo che si ottengono prodotti migliori, aziende più produttive e competitive sui mercati internazionali. In tutto questo noi programmatori abbiamo una grande responsabilità: quella di fare ottimi prodotti! Gli strumenti ci sono, adesso tocca a noi lavorare di tecnica e creatività.

Fabio Farnesi ffarnesi@edmaster.it

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

VISUAL STUDIO 2005 INCONTRA

MS Office

Sfrutta tutta la potenza di .NET ed aggiungi nuove funzioni a Word, Excel e Outlook

- ✔ **STRUMENTI:** Cosa sono e come funzionano i Visual studio Tools for Office
- ✔ **TECNICA:** Progetta un modello da usare per la creazione di fogli elettronici dinamici
- ✔ **TEORIA:** Dai Bookmark ai NamedRange ecco i controlli che pilotano i documenti
- ✔ **FUNZIONI AVANZATE:** Sviluppa un "add-in" che si integra nell'interfaccia delle applicazioni

pag. 14

FATTI LA SEGRETARIA PER LIVE MESSENGER

Ecco come si crea un plugin per MSN. In più: l'esempio completo su come programmare un bot che risponde anche quando tu non ci sei

pag. 62

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

http://forum.ioprogrammo.it

RIVISTA + CD-ROM in edicola

JBUILDER 2005 FOUNDATION EDITION

La soluzione Borland per lo sviluppo Java

Ha dell'incredibile questo IDE targato Borland. E' probabilmente l'unico vero ambiente RAD per lo sviluppo di applicazioni Java. Il suo concorrente potrebbe essere NetBeans ma per velocità e ricchezza di features, JBuilder rappresenta ancora la soluzione migliore per chi ha bisogno di un vero RAD per il linguaggio di SUN. L'IDE si presenta nel puro stile RAD di Borland, che come si sa è il capostipite del genere. Facile da utilizzare rappresenta il miglior modo di programmare in Java sia per chi inizia sia per l'utente esperto. La versioen foundation è completamente gratuita e priva di limitazioni. Un prodotto da tenere a portata di mano

Prodotti del mese

Phalanger 2.0

Il compilatore PHP per .NET

Si sa che .NET è una tecnologia che non dipende da un particolare linguaggio di sviluppo. Un'applicazione .NET viene compilata con una sorta di metacode per cui è possibile utilizzare un qualsiasi linguaggio ed ottenere applicazioni .NET. Phalanger implementa appunto il PHP per .NET. Facilmente integrabile in Visual Studio vi consentirà di sviluppare applicazioni PHP compilate in tecnologia .NET. I vantaggi sono notevoli, potrete infatti sfruttare gli innegabili vantaggi di un IDE evoluto come Visual Studio 2005. Ma non solo. In realtà potrete usufruire di tutti i vantaggi legati alla tecnologia .NET. I vostri siti infatti saranno compilati e usufruiranno dei vantaggi in termini di sicurezza e gestione della memoria legati ad IIS. Viceversa nella modalità standard PHP in ambiente Windows potrà girare solo come una normale ISAPI senza sfruttare i meccanismi interni di IIS

[pag.106]

JBOSS 4.0.4

L'Application Server per le imprese

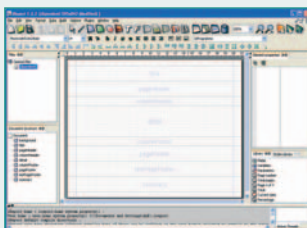
Volete sviluppare un'applicazione Web di livello Enterprise? Che faccia uso di J2EE ? Non potete prescindere da un Application Server. JBOss è il più usato dalle imprese che hanno bisogno di stabilità, sicurezza, affidabilità. Consente ovviamente di "hostare" applicazioni Java e ce ne parla ,collateralmente all'articolo su SEAM: il framework per Java Enterprise Edition, Ivan Venuti in questo stesso numero di ioProgrammo. J2EE rappresenta ormai una tecnologia matura ed ha evidenziato nel corso del tempo tutta la sua potenza. In termini di stabilità e sicurezza si tratta di una delle migliori tecnologie oggigiorno presenti sul mercato. Rimane qualche dubbio sulla velocità...

[pag.106]

Jasper Reports 1.2.5

Il creatore di Reports

Ce ne parla approfonditamente Federico Paparoni in questo stesso numero di ioProgrammo. Si tratta di uno strumento molto raffinato che consente di creare Report complessi partendo da una qualunque base di dati. E' facilmente integrabile in Java e leggendo l'articolo vi accorgerete che si tratta probabilmente dello strumento più sofisticato per la creazione di report da questo linguaggio, LA creazione di reportistica efficace è sempre stato uno dei maggiori problemi per il programmatore. Se da un lato la complessità della gestione dei dati segue una logica standard, dall'altro la capacità di presentare i dati in una forma aggregata comprensibile all'utente rappresenta uno scoglio non sempre facilmente superabile. E' nella semplicità d'uso che Jasper Reports fa sentire la sua potenza

[pag.107]

Struts 1.2.9

Il framework MVC per Java

Il pattern MVC è probabilmente il più usato da tutti gli sviluppatori in ogni linguaggio. Questo framework è il leader per quanto riguarda lo sviluppo di applicazioni Java secondo il pattern MVC. Molto comodo, stabile e affidabile, rappresenta la base di partenza per applicazioni che devono essere facilmente manutenibili. Uno standard da usare se progettate applicazioni di dimensioni generose, ma anche se volete sviluppare web application professionali. Struts è certamente un framework complesso e non immediatamente comprensibile, tuttavia una volta avviato un progetto e compresa la logica di base, vi renderete conto di quanto sia semplice effettuare un'update o una modifica a progetti anche complessi. E' infatti proprio in questo senso che struts trova i maggiori consensi

[pag.107]

I contenuti del libro

Imparare ASP.NET

Era nato qualche anno fa con la pretesa di soppiantare i vecchi linguaggi CGI . Inizialmente la sua struttura era quella classica dei linguaggi di Scripting. Con il tempo il linguaggio ASP si è trasformato in una completa tecnologia ed ha assunto la desinenza .NET per testimoniare il forte legame con l'omonimo framework di casa Microsoft. Oggi ASP.NET è una tecnologia solida e matura che funzione come spina dorsale di un enorme numero di applicazioni. Soprattutto ASP.NET trova collocazione in ambito aziendale la dove la solidità del framework su cui si basa può fare pesare tutto il proprio livello di eccellenza.
Antonio Pelleriti ci guida all'interno di questa tecnologia. Passo dopo passo apprenderete gli elementi che vi condurranno a diventare ottimi programmatori Web con ASP.NET

IL MANUALE FONDAMENTALE PER PROGRAMMARE SUBITO IL TUO PRIMO SITO WEB CON LE TECNOLOGIE MICROSOFT

- **Le basi per iniziare a programmare**
- **La struttura delle applicazioni**
- **I controlli lato server**
- **Le funzioni avanzate**

News

GOOGLE PER GLI SVILUPPATORI

Il gigante dei motori di ricerca ha appena messo a disposizione degli sviluppatori il sito "http://code.google.com". Il modello ricalca da vicino quello di SourceForge e consente agli utenti registrati con un account di Google di creare un nuovo progetto e renderlo disponibile ad un indirizzo predeterminato. A differenza di Sourceforge, lo strumento per il versioning del codice è il noto Subversion, e sempre a differenza di Sourceforge la modalità di ricerca è quella classica utilizzato dal gigante di Coupertino. Per chi è abituato ad utilizzare SourceForge, non sarà facile adattarsi alle tecnologie proposte da Google. È interessante tuttavia notare tuttavia quanto Google stia adottando una politica a tutto tondo che non tralascia di curare nessun particolare legato all'informatica. Non ci aspettavamo infatti che un gigante come SourceForge, cuore pulsante della community degli sviluppatori OpenSource potesse trovare un rivale proprio in Google che, notoriamente, sviluppa molto del suo software partendo proprio da progetti OpenSource

BRUTTO COLPO PER ZEND

Jani Taskinen, uno dei principali sviluppatori di PHP ha abbandonato il team di sviluppo in modo imprevisto e burrascoso. L'annuncio è stato dato sulla Mailing List ufficiale del progetto. Non sono state rese note le motivazioni ufficiali. Tuttavia indiscrezioni raccontano di dissidi dovuti a motivi politici relativi alle recenti posizioni assunte da Israele sul campo internazionale.
La sede principale di Zend è collocata infatti in territorio israeliano. Nonostante questo duro gesto, Zend non sembra avere avvertito eccessivi contraccolpi e di fatto continua parallelamente sia lo sviluppo di PHP che dello Zend Framework, senza contare gli altri prodotti come Zend Studio, collocandosi ai vertici delle classifiche per le software house più attiva

BORLAND RITORNA CON IL TURBO

L'annuncio è di quelli importanti: Borland, la storica software house pioniere nel mercato dei compilatori e degli ambienti di sviluppo RAD è tornata! Sembrava che dovesse rivolgersi esclusivamente al supporto e allo sviluppo di soluzioni per grandi clienti ed invece ecco arrivare una nuova fiammante divisione che si occuperà esclusivamente degli storici IDE e di tutto ciò che a che fare con il supporto agli sviluppatori. La prima mossa del nuovo Developer Tools Group di Borland è di quelle che contano. Non solo è previsto il rilascio di una nuova versione per ciascuno dei linguaggi compresi nel Borland Developer Studio, ma è anche previsto il lancio della nuova linea "Borland Turbo" che proporrà strumenti di sviluppo a basso costo per studenti, hobbysti, consulenti e professionisti. La nuova linea Turbo comprenderà: Turbo Delphi per Win 32, Turbo Delphi per .NET, Turbo C++ e Turbo C#. Di ciascun prodotto saranno disponibili due versione una denominata Turbo Explorer che sarà resa disponibile gratuitamente e scaricabile direttamente OnLine e una Turbo Professional che sarà commercializzata con un prezzo inferiore ai 500 Euro . L'uso del nome "Turbo" è significativo. Proprio con il Turbo Pascal, nella prima metà degli anni 90, Borland aveva rivoluzionato il mercato conquistando il cuore di milioni di programmatori e divenendo di fatto leader del mercato di sviluppo. A seguire il rilascio del primo Delphi, erede proprio di quel Turbo Pascal che

INTEL APRE ALL'OPEN SOURCE

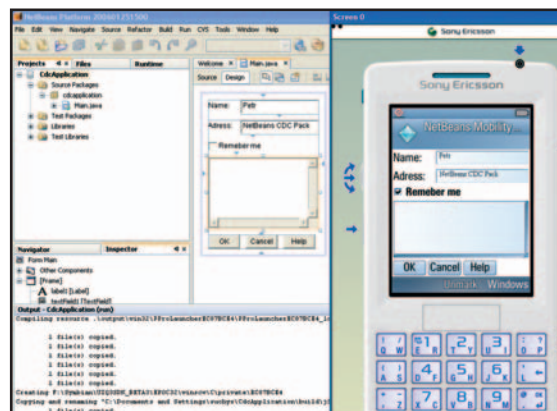
Intel non si è mai dimostrata ostile all'Open Source, ma l'annuncio delle ultime settimane risulta particolarmente importante. L'Intel Open Source Technology Center ha annunciato infatti la disponibilità del codice sorgente dei driver del chipset 965 Express.
L'annuncio è importante non solo perché questo chipset equipaggia un nutrito numero di schede grafiche ma anche perché proprio questo Chipset implementa funzionalità di rilievo relative al supporto DirectX e OpenGL. Altrettanto importante è l'intenzione di Intel di supportare lo sviluppo dei propri driver e in particolare di quelli rilasciati sotto licenza OpenSource attraverso un particolare sito: http://intellinuxgraphics.org/. Il nome del sito è tutto un programma: "Linux Graphics Driver from Intel Site", sinonimo di quanto Intel voglia supportare in ogni modo la comunità Open Source

così tanto aveva inciso sul mercato dei tool di sviluppo, segnava la strada verso gli IDE Rad. "La linea Turbo consente uno sviluppo software semplificato ma ricco di funzionalità grazie ad un tool di sviluppo user-friendly per la creazione e la distribuzione di applicazioni. Il rilascio di questa linea di prodotti non potrebbe giungere in un momento più adatto ed eccitante per noi: il Developer Tools Group si prepara in questi giorni ad intraprendere un'esperienza indipendente, unicamente focalizzata sulla creazione e miglioramento degli strumenti e della tecnologia per sviluppatori individuali e team" ha affermamo Francesca Ancona, responsabile per l'Italia del Developer Tools Group di Borland.

"Il brand è un classico, ma la tecnologia e le funzionalità presenti nei nuovi Turbo sono al massimo dell'avanguardia" dichiara Michael Swindell, Senior Director del Product Management di Borland Developer Tools Group.

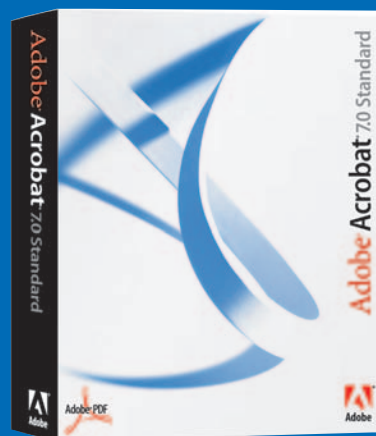
Noi di ioProgrammo non possiamo non dirci emozionati per questo importante ritorno che, siamo sicuri, vivacizzerà di molto il mercato dei tool di sviluppo

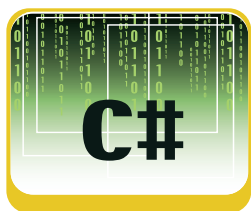
CODICE APERTO PER IL MOBILE SDK

Sun ha appena annunciato la disponibilità del codice sorgente del Netbeans Mobility Pack. Il codice in questione può essere scaricato direttamente dal sito http://mobility.netbeans.org/. L'annuncio è interessante. Il Netbeans Mobility Pack rappresenta un ottimo modo per scrivere applicazioni rivolte al crescente mercato di cellulari e palmari. All'interno del prodotto vengono integrati due profili: il Midp 2.0 e il CLDC. Il secondo si rivela particolarmente utile per piccoli dispositivi mobili dotati di connessione Internet, quali possono essere palmari e cellulari. La disponibilità del codice sorgente consente agli sviluppatori, da un lato di studiare le tecniche di interfacciamento con i dispositivi, dall'altro la personalizzazione del framework. L'esperienza insegna che questo genere di approccio, se da un lato da luogo ad una serie interminabile di framework non ufficiali, d'altra parte offre una spinta fuori dal comune allo sviluppo di estensioni e miglioramenti del framework

NUOVA RELEASE PER VISTA

È stata appena rilasciata ad un numero molto ristretto di beta tester la versione pre-RC1 del nascituro Windows Vista. Principalmente il team di sviluppo si è concentrato sulla velocità e sulla stabilità. Soprattutto l'esorbitante richiesta di risorse hardware ed in particolare di memoria, era stato il motivo centrale della preoccupazione del nutrito esercito di tester che aveva provato le precedenti Beta. La nuova versione sembra aver fatto significativi passi avanti proprio nel binomio prestazioni/stabilità. Il nuovo sistema operativo rappresenta un enorme punto di rottura con il passato e grazie alla vasta gamma di nuove funzionalità dovrebbe costituire anche terreno fertile per lo sviluppo di applicazioni di terze parti. Vista l'importanza del prodotto, il terrore di un flop dovuto all'eccessivo consumo di risorse hardware non aveva lasciato dormire sogni tranquilli agli sviluppatori. La nuova release sembra avere tranquillizzato il mercato che recepisce Windows Vista come un prodotto in grado di migliorare la produttività sia per il consumer sia per le aziende

PHP E PRODOTTI ADOBE SEMPRE PIÙ VICINI

Non si tratta di un prodotto ufficiale Adobe, ma di un interessante progetto portato avanti sul proprio blog da Mike Potter. Il prodotto è comunque di quelli che conta e merita una menzione particolare. Di fatto integra una serie di tecnologie che consentono di integrare il noto linguaggio per il Web con i più recenti strumenti di casa Adobe. Parliamo naturalmente di Flex o lo Spry Framework. Il codice del framework, come nella più classica delle tradizioni è sviluppato sotto licenza GPL ed è disponibile per il download all'indirizzo http://blogs.adobe.com/ mikepotter/adobe_php_sdk.zip. Singolare anche l'host scelto per lo sviluppo del prodotto. Si tratta infatti di uno dei primi tool di cui parliamo che fa uso del recente Google Project Hosting all'indirizzo http://code.google.com/hosting/. Ovvero il "SourceForge" di Google che consente appunto lo sviluppo di applicazioni in team

VISUAL STUDIO 2005 INCONTRA OFFICE

E SE IL CONTENITORE PRINCIPALE DI UNA NOSTRA APPLICAZIONE NON FOSSE UNA FORM MA AD ESEMPIO EXCEL O WORD? E' PROPRIO QUELLO CHE ACCADE QUANDO SI UTILIZZA VSTO. SCOPRIAMO COME ESTENDERE OFFICE IN MODO PROGRAMMATICO

Di cosa parleremo in questo articolo? Sostanzialmente della possibilità di sviluppare complete applicazioni in tecnologia .NET, la cui interfaccia verso l'utente non sia una normale Form, oppure il Browser, ma piuttosto l'interfaccia sarà costituita dalle normali applicazioni appartenenti alla Suite Microsoft Office!

Cerchiamo di spiegarci meglio. Siamo abituati a pensare a un documento Word come ad un elemento statico. Scriviamo al suo interno una lettera commerciale per esempio, e la salviamo in un'apposita directory. Non c'è una vera e propria interazione fra documento e utente. Un documento Word non è di per se un'applicazione attiva, piuttosto un soggetto passivo, un semplice contenitore "stupido" di dati. Proviamo a ipotizzare uno scenario diverso. Supponiamo che il nostro documento Word sia un'applicazione attiva! L'applicazione ad esempio che, dato il nome di un cliente e un codice, produce un output un contratto precompilato per la compravendita di una casa. E' solo un esempio. Ci sono milioni di applicazioni che possono sfruttare le potenzialità di Excel piuttosto che di Word. Nel nostro caso il documento Word conterrebbe due menu a tendina e un bottone. I due menu a tendina potrebbero essere utilizzati per fornire l'identificativo del cliente e il codice, mentre il bottone genererebbe il documento.

Dietro all'interfaccia utente fornita da Microsoft Word, potrebbe esserci il .NET framework, che da un lato fornisce gli stessi controlli che vengono utilizzati per le normali applicazioni Windows Form, dall'altro si occupa della connessione al database per il recupero dei dati e dell'interfacciamento con Microsoft Office. In mezzo a tutto questo, a disposizione del programmatore ci sarebbero linguaggi di alto livello come C# o Visual Basic. E ancora il documento Word potrebbe essere programmato direttamente dall'ambiente di Visual Studio. Come vedete siamo molto al di la della vecchia programmazione VBA.

COME INIZIARE?

Prima di tutto abbiamo bisogno di tre elementi fondamentali. Visual Studio 2005 almeno nella versione Standard Edition, Microsoft Office 2003 e i Visual Studio 2005 tools for Microsoft Office System. Il componente meno conosciuto è proprio quest'ultimo. Si tratta del framework da installare sopra Visual Studio per consentire la programmazione di Office. Chi ha un abbonamento MSDN può scaricarlo direttamente dal sito MSDN. Per gli altri la pagina di riferimento è http://msdn.microsoft.com/vstudio/products/vsto/default.aspx. Inutile dire che il prodotto ha un costo elevato, ma se siete programmatori all'interno di un'azienda, è molto probabile che uno strumento del genere sia in grado di innalzare la produttività aziendale in modo esponenziale

UN PRIMO ESEMPIO

Prima di addentrarci nella teoria sarà utile fornire un esempio semplice che lasci intuire quali sono le potenzialità di VSTO. Dopo aver installato il prodotto, lanciando Visual Studio, troverete a disposizione diversi nuovi tipi di template, tutti raggruppati sotto l'etichetta "Office". Scegliamo di creare un'applicazione "Documento di Word" e proseguiamo

Ci verrà richiesto se vogliamo creare l'applicazione partendo da un modello Word esistente, oppure da un modello Word creato da zero. Scegliamo questa seconda opzione e proseguiamo

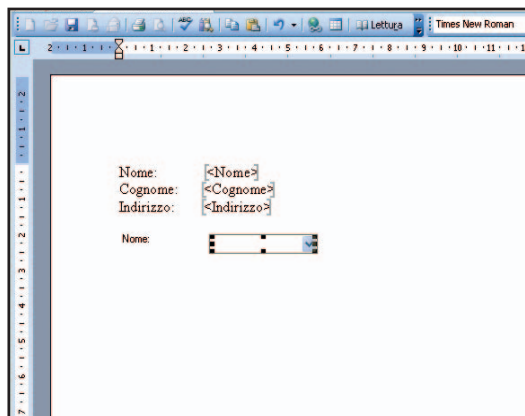
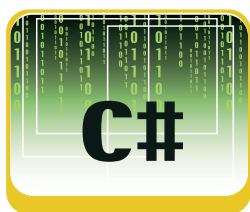
A questo punto entrano in ballo le politiche di sicurezza. Office deve poter accedere al .NET framework. Questo passo è importante, perché lascia intuire alcune delle differenze fondamentali con il precedente utilizzo delle macro tramite VBA. In questo nuovo scenario è il .NET framework a gestire l'intera sicurezza dell'applicazione. Rispondiamo OK per consentire ad office di poter interagire con gli strumenti di sistema e proseguiamo

Il risultato di questa serie di operazioni è quello che vedete in figura. Il documento di Word è diventato un contenitore per un'applicazione progettata con Visual Studio!

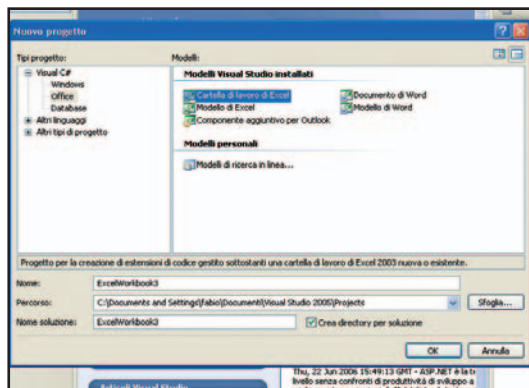
Come potete notare dall'immagine, l'intero documento Word vive all'interno di Visual Studio. Potete scrivere al suo interno se volete, ma anche trascinarvi componenti dalla toolbar e programmarli in Visual Basic o C#. Proviamo ad esempio ad aggiungere alla nostra applicazione Word un bottone, e a programmarne l'evento OnClick come segue:

```
private void button1_Click(object sender,
                                        EventArgs e)
    {
        MessageBox.Show("Un'applicazione con
    Word", "Dialog di prova",MessageBoxButtons.OK,
                          MessageBoxIcon.Information);
    }
```

Se provate a compilare questa prima applicazione, otterrete non un eseguibile ma un documento Word. Se lo aprite, scoprirete al suo interno il bottone, e cliccando sul bottone otterrete una dialog box esattamente come avevamo previsto nella gestione dell'evento OnClick da noi programmata.
Naturalmente se lanciate l'applicazione dall'interno di Visual Studio, verrà nuovamente aperto Microsoft Office, ma questa volta avrete a disposizione tutti gli strumenti di debugging tipici del vostro ambiente di programmazione.
Un altro esempio interessante è quello relativo all'utilizzo del database. Non ci dilunghiamo su quelle che sono le modalità di connessione e reperimento dei dati. La tecnica è identica a quella classica utilizzata in Visual Studio, ovvero aggiungere una fonte di dati, seguire il wizard, inserire i campi nel documento. Potete dare uno sguardo all'immagine seguente:

Aprendo questo file .doc si otterrà un'applicazione che scorre gli elementi di un database tramite un menu a tendina e riempie i campi corrispondenti Nome, Cognome e indirizzo.

FACCIAMOLO CON EXCEL

La potenza di VSTO risiede soprattutto nel fatto che è possibile programmare qualunque tipo di applicazione Office utilizzando le tecniche di Visual Studio. Tuttavia è anche vero che ciascun software appartenetene alla suite di Office è dotato di funzionalità particolari, per cui non si può prescindere dall'utilizzare colonne, righe e celle se si usa Excel. Per questo motivo quando viene creata una nuova applicazione di tipo Excel, compaiono due nuovi componenti che servono appunto a gestire gli elementi tipici di excel, ovvero il componente NamedRange e il componente ListObject. Prima di procedere con qualche esempio e la descrizione di questi due oggetti, sarà il caso di iniziare dal più classico degli elementi di un foglio di lavoro di Excel, ovvero la cella.

Il metodo più semplice per lavorare con una cella è identificarla con un nome. Per farlo è sufficiente selezionarla e digitarne in nome nella casella che la rappresenta, così come rappresentato dalla figura seguente:

Accedervi è poi semplicissimo, è sufficiente utilizzare una sintassi di questo tipo:

```
private void button1_Click(object sender,
                                   EventArgs e)
    {
        lamiacella.Value = 10;
    }
```

Notate che per il nostro esempio abbiamo anche utilizzato un bottone e ne abbiamo gestito l'evento OnClick. Ovviamente il bottone in questione non è altro che un componente del .NET framework.

E' possibile accedere al valore della cella anche come segue:

```
private void button1_Click(object sender,
                                   EventArgs e)
    {
        Globals.Foglio1.Range["A6",missing].Value2
                                          = 10;
    }
```

Notate però che in questo caso abbiamo fatto uso di un metodo Range, che accetta due valori. Nel nostro caso gli abbiamo passato un valore di cella e un secondo valore vuoto, per identificare che vogliamo lavorare sulla cella di coordinate A6, ma avremmo anche potuto referenziare un intero Range di celle.

Proviamo a verificare il seguente esempio:

```
private void button1_Click(object sender,
                                   EventArgs e)
    {
        Globals.Foglio1.Range["A6","A10"].Value2 =
                                          10;
    }
```

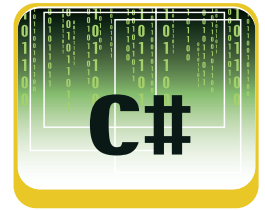
Noterete che adesso il valore viene assegnato ad un intero range di celle. Possiamo anche definire un nome per un Range, come segue:

```
private void button1_Click(object sender,
                                   EventArgs e)
    {
this.Controls.AddNamedRange(this.Range
                    ["A6","A10"], "namedRange1");
        namedRange1.Value = "10";

    }
```

Notate che però proprio in quest'ultimo esempio abbiamo introdotto un nuovo elemento ovvero il "NamedRange". Come potete constatare abbiamo aggiunto a runtime al foglio di lavoro un nuovo oggetto di tipo NamedRange definito dall'interevallo di celle A6:A10 e identificato dal nome namedRange1. Avremmo ottenuto lo stesso identico effetto trascinando sul foglio di lavoro un controllo di tipo NamedRanged direttamente dalla barra degli strumenti.

Notate che un namedRange dispone di proprietà ed eventi particolari. Supponiamo di aver creato un controllo di tipo NamedRange avendolo trascinato dalla toolbar sul foglio di lavoro e che questo oggetto si chiami MyNameRange. Osservate il seguente codice:

```
private void button1_Click(object sender,
                                   EventArgs e)
    {
        MyNameRange.Value = 10;
    }
private void
 MyNameRange_Change(Microsoft.Office.Interop.Excel
                                  .Range Target)
    {
```

```
        string cellAddress =
                     Target.get_Address(missing,
missing,Microsoft.Office.Interop.Excel.XlReferenceStyl
                     e.xlA1,missing, missing);
        MessageBox.Show("Le celle " + cellAddress
                     + " hanno cambiato valore.");
    }
```

Nel momento in cui l'utente effettua un click sul bottone, viene aggiornato il valore di un intervallo di celle, immediatamente si scatena l'evento "Change" per "MyNameRange", e viene visualizzata una dialog box contenente l'elenco delle celle il cui valore è cambiato.

MANIPOLARE I DATI

Un primo esempio leggermente più complesso dei precedenti e che mostra le potenzialità della programmazione con VSTO è quello che mostreremo da qui a breve. In particolare creeremo un dataset statico, e binderemo i valori in esso contenuti ad un NamedRange, utilizzeremo poi un bottone per scorrere i vari dati sempre all'interno di un'unica cella.
L'esempio è il seguente:

```
public partial class Foglio1
{
    private string[] Redattori = { "Fabio",
        "Domenico", "Luca", "Anna","Rossana" };
    private DataSet ds;
    private void Foglio1_Startup(object sender,
                         System.EventArgs e)
    {
        ds = new DataSet();
        DataTable table =
                  ds.Tables.Add("Redattori");
        DataColumn column1 = new
             DataColumn("Nomi", typeof(string));
        table.Columns.Add(column1);
        // Riempie le righe della tabella
        // Con i valori presi dall'array redattori
        DataRow row;
        for (int i = 0; i < Redattori.Length; i++)
        {
            row = table.NewRow();
            row["Nomi"] = Redattori[i];
            table.Rows.Add(row);
        }
        // Crea un "bind" fra la property Value2 del
        // NamedRanged e il contenuto della tabella
        // nomi
        // Dell'array redattori
        Binding binding1 = new Binding("Value2",
                  ds,"Redattori.Nomi", false);
        MyNameRange.DataBindings.Add(binding1);
    }
    private void button1_Click(object sender,
                                 EventArgs e)
    {
        if (MyNameRange.BindingContext != null)
        {
            BindingManagerBase bindingManager1
        =MyNameRange.BindingContext[ds, "Redattori"];
            // Mostra il contenuto del prossimo
                                          record
            if (bindingManager1.Position <
                     bindingManager1.Count - 1)
            {
                bindingManager1.Position++;
            }
            // Mostra il primo record
            else
            {
                bindingManager1.Position = 0;
            }
        }
```

Il codice è autoesplicativo ed è molto simile a quello proposto da Microsoft nella documentazione ufficiale MSDN.
Prima di tutto abbiamo dichiarato alcuni campi privati che abbiamo utilizzato nel resto del procedimento. In particolare

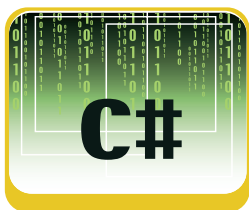
UTILIZZARE LE FORMULE

Considerato "Risultato" come una cella con nome, il seguente esempio esegue la sommatoria dei valori presenti in un range

```
private void button1_Click
        (object sender, EventArgs e)
{
    risultato.Formula =
                  "=SUM(A5:A10)";
    risultato.FormulaHidden =
                            true;
    risultato.Calculate();
}
```

```
ds = new DataSet();
        DataTable table =
                  ds.Tables.Add("Redattori");
        DataColumn column1 = new
             DataColumn("Nomi", typeof(string));
        table.Columns.Add(column1);
```

Crea un nuovo oggetto di tipo DataSet. All'interno del DataSet crea una tabella di tipo DataTable e dal nome redattori. Questa tabella ha una colonna di tipo DataColumn che verrà referenziata dal puntatore "Nomi".

```
        DataRow row;
        for (int i = 0; i < Redattori.Length; i++)
        {
```

```
            row = table.NewRow();
            row["Nomi"] = Redattori[i];
            table.Rows.Add(row);
        }
```

Ciascuna riga della tabella in questione viene riempita con il contenuto dell'array redattori.

```
Binding binding1 = new Binding("Value2",
                        ds,"Redattori.Nomi", false);
        MyNameRange.DataBindings.Add(binding1);
    }
```

A questo punto viene creato un oggetto di tipo Binding e collegato che effettua fisicamente il collegamento fra il NamedRange e il dataset.
Leggermente più complessa la gesione dell'evento OnClick, ma semplicemente perché fa uso di un bindingmanager per spostare il cursore da un dato all'altro. In ogni caso al di la della sintassi leggermente complessa, la semantica è davvero semplice.

COLLEGAMENTO AD UN DATABASE

L'esempio proposto nell'articolo è perfettamente replicabile anche quando il dataset viene riempito da dati provenienti, ad esempio da SQL Server. Tutto quello che dovete fare è seguire il solito wizard per il collegamento al database. A questo punto è sufficiente "agganciare" il NamedRange all'apposito BindingManager creato dal Wizard, il codice di gestione dell'evento OnClick rimane quasi identico al precedente

```
private void
        button1_Click(objectsender,
                        EventArgs e)
    {
        if
    (MyNameRange.BindingContext !=
                            null)
        {
            if
    (redattoriBindingSource.Position <
    redattoriBindingSource.Count - 1)
            {
    redattoriBindingSource.Position++;
            }
            else
            {
    redattoriBindingSource.Position = 0;
            }
        }
```

L'OGGETTO LISTOBJECT

Accanto al NamedRange, VSTO propone un nuovo controllo dalle caratteristiche sufficientemente complesse. Si tratta appunto del ListObject. Questo tipo di oggetto può, per certi versi, essere assimilato ad una DataGrid. Consente di rappresentare i dati in forma gabellare utilizzando la griglia composta da celle e righe proposta da Excel. L'azione più semplice che si può compiere con un ListObject è quello di trascinare quest'oggetto su un foglio Excel. Vi verrà chiesto di selezionare un'area di rappresentazione dell'oggetto. Ovvero quali celle dovranno contenere i dati?
Inserito così in modo semplice, l'oggetto in questione restituisce una DataGrid vuota, se lo si vuole agganciare ad un DataSet è sufficiente agire sulla proprietà "DataSource", così come mostrato in figura:

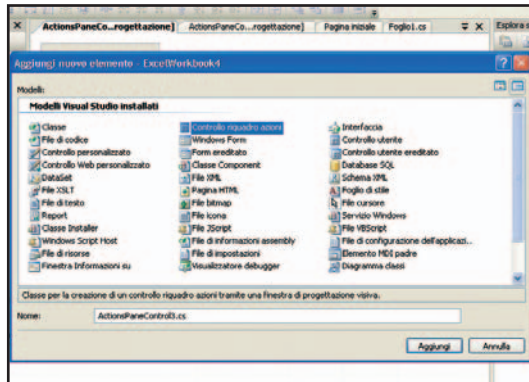
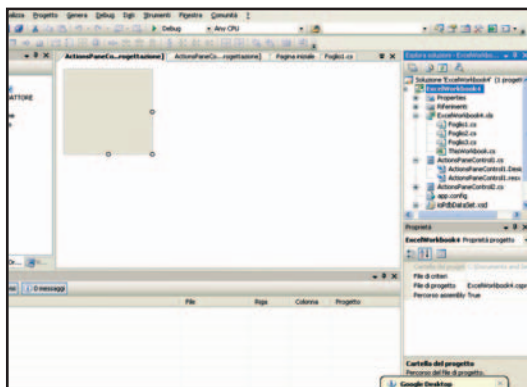
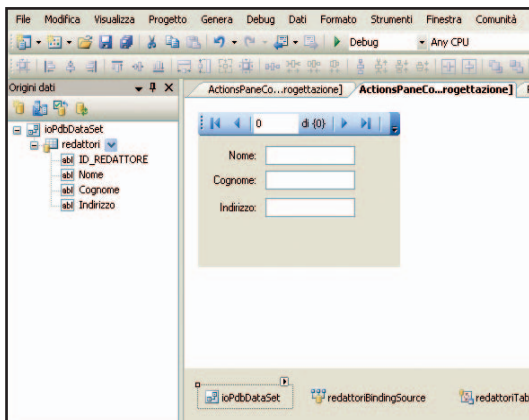
Una volta ottenuto l'elenco dei dati, non ci rimane che aggiungere la funzione di modifica/insert dei dati sul server, per farlo utilizzeremo un'altra tecnica, ovvero quella degli ActioPane

GLI ACTIONPANE

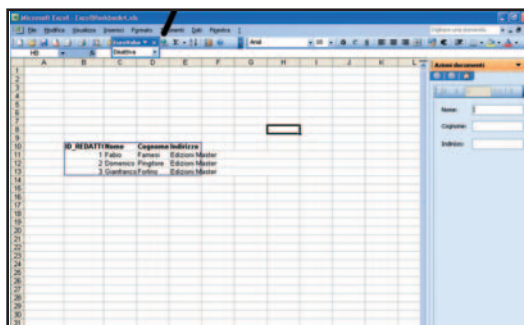
Possiamo pensare ad un Action Pane come ad uno dei riquadri che tipicamente riempiono la parte destra di Microsoft Word o Excel. Tipicamente un Action Pane può interagire con il documento che lo espone, ad esempio per fornire funzioni avanzate. Un Action Pane può essere programmato come una normale Windows Form. I passi per programmare un Action Pane sono i seguenti:

- Inserire nel progetto uno User Control
- Eseguire la programmazione dello User Control secondo le proprie specifiche
- Caricare lo User Control all'interno del documento tramite l'evento Startup del documento

Il primo passo è facile, è sufficiente portarsi nel solution exporer e cliccare con il tasto destro del mouse alla voce "aggiungi nuovo elemento"

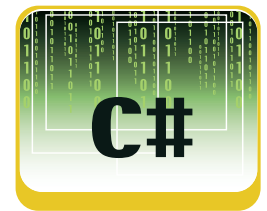
Nella dialog box successiva selezioniamo "Controllo Riquadro Azioni" (Action Pane per chi usa Visual Studio 2005 in inglese)

A questo punto disporrete di una normale form di progettazione sulla quale potete inserire i vostri controlli.

Nel nostro caso vi abbiamo inserito i controlli di gestione per il database, esattamente come avremmo fatto in una normale Windows Form

A questo punto non ci rimane che gestire l'evento startup del Foglio per fare in modo che l'action pane venga attivato e visualizzato in corrispondenza all'avvio dell'applicazione. Anche questa operazione non ha una complessità particolarmente elevata

```
using Microsoft.Office.Tools;
using Microsoft.Office.Tools.Excel;
[...]
public partial class Foglio1
    {
        UserControl uc = new ActionsPaneControl1();
        private void Foglio1_Startup(object sender,
                                    System.EventArgs e)
        {
Globals.ThisWorkbook.ActionsPane.Controls.Add(uc);
            Globals.ThisWorkbook.ActionsPane.Visible =
                                                true;
[...]
```

Tutto quello che abbiamo fatto è stato aggiungere il nostro personale Action Pane al Foglio di lavoro e renderlo visibile tramite l'apposita proprietà. Infine nell'immagine sottostante potete visualizzare il risultato del nostro lavoro

DEPLOYMENT DELL'APPLICAZIONE

Il deployment di un'applicazione sviluppata con VSTO rappresenta probabilmente la parte più delicata di tutta l'architettura. Una prima soluzione è rappresentata dall'uso di ClickOnce, la nuova tecnologia di Microsoft che consente di pacchettizzare e upgradare i pacchetti in modo semplice. Non ne parleremo in modo approfondito in questo articolo, ma dedicheremo a breve a ClickOnce una puntata specifica. In ogni caso sul client devono essere presenti nell'ordine:

- Il documento programmato con VSTO
- Il codice e gli assembly legati al documento
- Il framework .NET 2.0
- Una versione Professional di Office 2003 oppure una versione Standalone di Excel/ Word 2003
- Il runtime di VSTO

L'ultimo passo importante per il deployment è l'impostazione delle policy di security del sistema. Ma di questo parleremo nei prossimi numero di ioProgrammo

DATI SUL WEB SEMPRE CORRETTI

RIEMPIRE UNA FORM È UNA DELLE OPERAZIONI PIÙ FREQUENTI CHE SI POSSONO COMPIERE NAVIGANDO SU INTERNET. MA COME GARANTIRE CHE GLI UTENTI NON SCRIVANO DATI CASUALI NEI VARI CAMPI? ECCO COME ASP.NET RISOLVE IL PROBLEMA DELLA VALIDAZIONE

Una delle caratteristiche più diffuse dei siti web, che ormai non sono più mere statiche vetrine ma applicazioni vere e proprie eseguite nel browser, massiccia è la presenza di campi da compilare in maniera da ricevere informazioni dall'utente, per una qualche motivazione. Possono essere moduli per la registrazione, per richiedere dati su cui effettuare un calcolo, e così via.
Ma quando una pagina web consente al visitatore di immettere dei dati, per esempio per riempire i campi di un modulo, con il proprio nome e email, o per scegliere la propria nazionalità da una casella combinata a discesa, è importante verificare che i dati siano stati effettivamente inseriti ed anche nel formato corretto. Il termine corretto in questo caso non è univoco, il classico esempio è quello dell'indirizzo email, il cui formato è ben noto, e quindi abbastanza facile capire come verificare, anche se poi in pratica il discorso è tutt'altro che semplice.
ASP.NET, già dalle versioni precedenti, mette a disposizione dei controlli standard per effettuare questo genere di verifiche: essi prendono il nome di validator control.

I CONTROLLI DI VALIDAZIONE

Ogni controllo di validazione o validatore, deriva da una classe **BaseValidator**, che a sua volta deriva dalla classe **Label**, cioè dalla classica etichetta per visualizzare un testo, eventualmente formattato. In effetti l'obiettivo finale dell'utilizzo di un controllo di validazione è quello di mostrare un'informazione, tramite un'etichetta di testo, che illustri se e quale errore si sia verificato nel compilare i campi di una pagina, o anche in qualche operazione avvenuta lato server.
Così, ogni controllo validatore può riferirsi ad un controllo di input, per esempio una **TextBox**, una **DropDownList**, e così via, ed ognuno dei controlli di input potrà essere affidato alla verifica da parte di più controlli di validazione.
Per esempio, nel caso in cui la TextBox debba contenere un indirizzo e-mail e questo debba essere obbligatoriamente inserito, si potrebbe associare alla casella di testo sia un controllo che verifichi la validità del formato, ed un secondo che verifichi l'avvenuta compilazione del campo.
La classe **Page**, da cui una classica pagina web in ASP.NET deriva, fornisce una proprietà **Validators**, che restituisce la collezione di tutti i controlli validatori presenti nella pagina stessa.
I controlli presenti nella collezione possono essere poi validati tutti in un sol colpo, ad esempio quando si fa clic su un pulsante per inviare i dati di un modulo al server, oppure singolarmente. Per gestire il primo caso, la classe Page fornisce infatti un metodo **Validate**, metodo posseduto anche da ogni controllo derivato da **BaseValidator**. Quest'ultima classe mette poi a disposizione una proprietà **IsValid**, che restituisce un valore booleano indicante se il controllo contiene dati validi, impostato appunto dopo la chiamata al metodo **Validate**.

ASSOCIARE UN VALIDATORE

Per associare un controllo di validazione ad un controllo di input, la classe BaseValidator fornisce la proprietà **ControlToValidate**. La proprietà deve essere impostata utilizzando il valore della proprietà ID del controllo destinatario della validazione.
Naturalmente non tutti i controlli web di ASP.NET 2.0 possono essere associati ad un controllo validatore, perché in molti casi non avrebbe senso. Le classi che è possibile validare sono identificati dall'attributo ValidateProperty, e fra questi quelli tipicamente utilizzati sono i controlli **TextBox**, **ListBox**, **DropDownList**, **File Upload**, **RadioButtonList**, mentre ad esempio non sono associabili **ad un controllo** validatore i **controlli CheckBox**, **RadioButton**, **CheckBox List**. Ciò non significa che questi ultimi non sono veri ficabili, ma si dovrà fare del lavoro in più, come vedremo, per mezzo di un validatore personalizzato.

UNA PAGINA DI REGISTRAZIONE

Per provare i controlli validatori sul campo, si creerà una pagina Asp.net, contenente un classico modulo di registrazione, ed il cui aspetto finale assomiglierà a quello in figura 1.

Fig. 1: ***La pagina di registrazione***

Essa contiene i classici campi da compilare in un modulo di registrazione. Di questi, si vuol renderli obbligatori tutti tranne il campo "Anno di nascita", che è un'informazione aggiuntiva. Inoltre si vuole che il campo "Username" venga compilato con una stringa lunga per esempio da 5 a 15 caratteri, il campo "Password" dovrà essere confermato dal seguente campo di conferma, ed il campo "Email" dovrà contenere una stringa nel classico formato "nome@dominio.est".
In tutti questi compiti i controlli di validazione ci daranno una mano, anzi, a dire la verità , praticamente faranno quasi tutto da soli.

VERIFICARE I CAMPI OBBLIGATORI

Quando è necessario rendere obbligatoria la compilazione di un campo di input, giunge in aiuto allo sviluppatore il controllo validatore **RequiredFieldValidator**.
Il primo passo è creare il controllo aggiungendolo alla pagina, in corrispondenza del punto in cui si vuol visualizzare il messaggio, trascinandolo dalla barra degli strumenti di Visual Studio, oppure inserendo nel codice HTML il tag corrispondente. Per esempio per controllare che il campo Username sia stato compilato si potrà scrivere il codice seguente:

```
<td>
<asp:TextBox ID="txtUsername"
                runat="server"></asp:TextBox>
</td>
<td>
<asp:RequiredFieldValidator
        ID="RequiredFieldValidator1" runat="server"
ControlToValidate="txtUsername"
                ErrorMessage="Username
                                obbligatoria">
</asp:RequiredFieldValidator>
</td>
```

Il controllo di input di cui verificare l'avvenuta compilazione, è indicato tramite la proprietà **ControlToValidate**. La proprietà **ErrorMessage** serve invece ad impostare il messaggio da visualizzare nel casi in cui si verifichi un errore, e cioè in questo caso la mancata presenza di una stringa nel campo **txtUsername**, Se al posto della proprietà **ErrorMessage**, o in concomitanza ad essa, si imposta la proprietà **Text**, il valore di quest'ultimo verrà visualizzato in ogni caso all'esecuzione della pagina. Così facendo sarà possibile impostare un valore che indichi all'utente di inserire lo **username**, per esempio un asterisco, come si vede spesso, mentre il valore di **ErrorMessage** potrà essere visualizzato in un riepilogo definito tramite un **Validation Summary** che si vedrà in seguito. Facendo lo stesso per tutti i controlli da rendere obbligatori, e mandando in esecuzione la pagina, si provi a premere il pulsante Invia senza compilare alcuno dei campi prima, e compilandone qualcuno poi. I campi non compilati saranno contrassegnati dal testo inserito come proprietà Error Message oppure Text, come in figura 2.

Registrazione
Compila i campi seguenti:
Username: *
Password: *
Conferma password:
Email: *
Anno di nascita:
Invia

Fig. 2: ***I campi obbligatori della pagina***

CONFRONTARE DUE CAMPI

Il controllo **CompareValidator** consente di eseguire un confronto fra i valori inseriti in due campi, oppure fra il contenuto di un campo ed un altro valore costante secondo diversi operatori.
Il seguente codice mostra come aggiungere un **CompareValidator** per verificare che una **TextBox** contenga un valore numerico minore dell'anno attuale:

```
<asp:CompareValidator ID="CompareValidator1"
                                    runat="server"
ControlToCompare="txtPassword"
          ControlToValidate="txtConfermaPassword"
ErrorMessage="I campi password non coincidono">
</asp:CompareValidator>
```

Innanzitutto è necessario indicare il controllo da validare, mediante il suo ID, in questo caso **txtConfermaPassword**, il controllo con cui eseguire il confronto è invece determinato dalla proprietà **ControlToValidate**, quindi la proprietà **ErrorMessage** contiene la stringa da mostrare nel caso di valore inserito non valido.
Se invece si vuol eseguire un controllo fra il contenuto di un campo ed una costante, si possono utilizzare diverse proprietà. Si supponga di voler verificare che il campo Anno di nascita, se compilato, contenga un valore innanzitutto numerico, e che sia naturalmente minore dell'anno corrente.
Il tipo di confronto da eseguire viene determinato dalla proprietà Operator, che può essere impostata ad uno dei valori dell'enumerazione ValidationCompareOperator, i cui membri definiscono i classici operatori di confronto. Come nel caso precedente, esso sarà per default considerato come Equal, cioè verrà fatto un confronto di uguaglianza.
In questo caso si è scelta la possibilità di eseguire il controllo da codice, e piuttosto che impostare una costante 2006, si ricaverà l'anno attuale dalla data del sistema, impostando così la proprietà **ValueToCompare**. Trattandosi di un valore intero si imposta inoltre la proprietà Type al valore **ValidationDataType.Integer**:

```
protected void Page_Load(object sender, EventArgs
                                                e)
{
     YearCompareValidator.ValueToCompare =
                       DateTime.Today.Year.ToString();
     YearCompareValidator.Type =
                          ValidationDataType.Integer;
     YearCompareValidator.Operator =
                  ValidationCompareOperator.LessThan;
}
```

VERIFICARE INTERVALLI DI VALORI

Il controllo RangeValidator consente di verificare che il valore inserito in un controllo sia compreso in un intervallo di valori predeterminato oppure calcolato a runtime. Nel primo caso si può impostare tutti in maniera dichiarativa. Per esempio per verificare che il campo Anno contenga un valore positivo e minore di 2006, basterà aggiungere il seguente codice di markup:

```
<asp:TextBox ID="txtAnno"
                         runat="server"></asp:TextBox>
<asp:RangeValidator ID="RangeValidatorAnno"
                                      runat="server"
ControlToValidate=" txtAnno"
                               ErrorMessage="Inserire
      un valore fra 0 e 2006" MaximumValue="2006"
                   MinimumValue="0" Type="Integer">
</asp:RangeValidator>
```

Anche in questo caso viene impostato il tipo Integer mediante la proprietà Type, mentre l'intervallo di valori consentito è determinato dai valori dati a MinimumValue e MaximumValue.
La stessa cosa può naturalmente essere realizzata da codice, impostando come desiderato le proprietà, per esempio:

```
RangeValidatorAnno.MaximumValue=
                                    Datetime.Today.
                                    Year.ToString();
```

ESPRESSIONI REGOLARI

In alcuni casi è necessario verificare che l'input dell'utente rispetti un certo formato, e nell'esempio è presente un campo Email, che è proprio da inserire con un formato ben definito. Le espressioni regolari in tali occasioni sono la soluzione ideale, ed il controllo Regular ExpressionValidator, automatizza la verifica del formato inserito in un campo, utilizzando un'e-

SE NON HO IIS INSTALLATO?

Anche senza IIS è possibile utilizzare il codice in allegato. Basta decomprimerlo in una cartella, e da Visual Studio 2005 o Visual Web Developer aprirlo tramite la voce Open -> Web Site del menù File, quindi scegliere File System e posizionarsi sulla cartella scelta. In questo modo sarà possibile eseguire la pagina di esempio con il server di sviluppo e senza IIS.

spressione regolare impostata tramite la proprietà ValidationExpression:

```
<asp:TextBox ID="txtEmail"
        runat="server"></asp:TextBox>
<asp:RegularExpressionValidator
        ID="EmailRegularExpressionValidator"
        runat="server"
    ControlToValidate="txtEmail"
        ErrorMessage="indirizzo non valido"
    ValidationExpression="\w+([-
    +.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

Visual Studio .NET 2005 permette di impostare alcune delle espressioni regolari più comuni scegliendole da un elenco, come mostrato in figura 3.

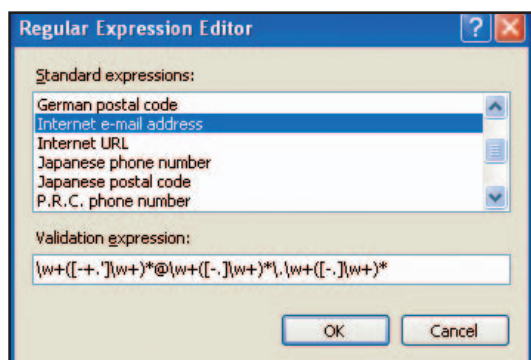



Fig. 3: ***Verificare un indirizzo email con una espressione regolare***

L'espressione regolare per la validazione dell'e-mail appena utilizzata ne è un esempio, anche se certamente non rispetterà tutti i formati possibili, facendo passare magari degli indirizzi non validi, ma nella maggioranza dei casi può andare bene.

VALIDAZIONE PERSONALIZZATA

Se nessuno dei precedenti controlli validatori fornisse le funzionalità ricercate, o semplicemente se si vuol validare un controllo che non è marcato con l'attributo ValidationProperty, come una CheckBox, è possibile utilizzare un controllo CustomValidator. Supponiamo come detto che il campo Username debba contenere una stringa di lunghezza minima 5 caratteri e massima di 15.

Per eseguire tale verifica il controllo CustomValidator consente di utilizzare sia un metodo lato client, scritto per esempio in Javascript, sia un metodo di validazione eseguito sul server.

Nel primo caso è necessario impostare il nome della funzione da eseguire nella proprietà ClientValidationFunction:

```
<asp:CustomValidator ID="CustomValidator1"
    runat="server" ControlToValidate="txtCodice"
        ErrorMessage="Codice non valido"
            ValidateEmptyText="True"
        ClientValidationFunction="ControllaCodice">
</asp:CustomValidator>
```

In questo caso verrà eseguita la funzione javascript ControllaCodice, mentre la proprietà ValidateEmptyText indica che il contenuto deve essere verificato anche se il campo fosse vuoto. La funzione indicata deve rispettare una particolare firma, di cui il seguente è un esempio valido:

```
<script type="text/javascript"
            language="javascript">
    function ControllaLunghezzaUsername(source,
                                args)
    {
        if(args.Value.length >4 &&
                args.Value.length <16 )
            args.IsValid=true;
        else args.IsValid=false;
    }
</script>
```

Deve dunque essere presente un parametro source, ed un parametro args da utilizzare per ricavare il valore del campo e per impostare la validità per mezzo della proprietà IsValid.

Dal lato del server, si può utilizzare un'analoga controparte, anche contemporaneamente alla funzione lato client. Basta gestire l'evento ServerValidate:

```
protected void
    UsernameLengthValidator_ServerValidate(object
            source, ServerValidateEventArgs args)
  {
    if (args.Value != null)
    {
        int len = args.Value.Length;
        args.IsValid = (len > 4 && len < 16);
    }
  }
```

Il parametro args è corrispondente a quello già visto nella funzione JavaScript, e si utilizza in maniera perfettamente equivalente.

È anche possibile tralasciare di impostare la proprietà ControlToValidate e quindi ricavare lato server i valori contenuti in tutti i controlli che si vuole, dato che in tal caso la proprietà

"Programming ASP.NET 2.0 – Core Reference", Dino Esposito (Microsoft Press)
"Professional ASP.NET 2.0", Evjen et al. (Wrox Press)

Value del parametro args sarà uguale ad una stringa vuota. In quest'ultimo caso è dunque possibile validare contemporaneamente più campi e come detto validare anche controlli non marcati con l'attributo ValidationProperty, come un CheckBoxList, una CheckBox, o un RadioButton.
Bisogna però tenere presente che il metodo ServerValidate verrebbe eseguito dopo un PostBack, e quindi solo nel caso in cui non ci siano altri errori gestiti da altri validatori nella pagina a bloccare tale PostBack.

L'AUTORE

Antonio Pelleriti, ingegnere informatico, sviluppa software da più di dieci anno e si occupa di .NET sin dalla prima versione Beta. È content manager della .NET community www.dotnetarchitects.it. Può essere contattato per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@ioprogrammo.it

RIEPILOGARE GLI ERRORI

Il controllo ValidationSummary permette di riepilogare in una singola Label tutti gli eventuali errori trovati dai controlli di validazione di una pagina. È possibile mostrare la lista di errori in differenti modi, definiti dall'enumerazione ValidationSummaryDisplayMode, impostando la proprietà DisplayMode del controllo. Per default essa è pari al valore BulletList, cioè viene utilizzato un elenco puntato.
È possibile anche fornire un testo di intestazione, mediante la proprietà HeaderText.
Oltre alla Label sulla pagina, in corrispondenza del ValidationSummary, è possibile visualizzare una MessageBox con lo stesso riepilogo, impostando la proprietà ShowMessage Box.

```
<asp:ValidationSummary ID="ValidationSummary1"
    runat="server" HeaderText="Riepilogo degli errori"
                        ShowMessageBox="True" />
```

Il riepilogo degli errori, cioè il controllo ValidationSummary, viene visualizzato solo se c'è almeno un errore nella pagina e se la proprietà ShowSummary non è stata impostata a false.

VISUALIZZAZIONE STATICA O DINAMICA

Ogni controllo di validazione possiede la proprietà Display, che può assumere, i valori Static e Dynamic, e None. Il primo caso, lo spazio per la Label contenente il messaggio di errore viene riservato sulla pagina, ed occupato sia che l'errore si verifichi sia nel caso contrario. Viceversa, con Dynamic, il calcolo dello spazio da allocare sulla pagina viene fatto dinamicamente. Ciò è di fondamentale importanza quando si utilizzano più controlli di validazione affiancati, in quanto nel caso si verifichi solo uno degli errori, e magari non il primo, la Label che mostra l'errore apparirebbe in mezzo alla pagina, lasciando uno sgradevole spazio bianco riservato staticamente ad un altro errore. Il valore None naturalmente non mostrerà il messaggio anche in caso di errore.

Combinando le proprietà ErrorMessage e Text dei controlli validatori, ed un controllo ValidationSummary, è possibile così costruire diversi scenari.
Per esempio si può impostare contemporaneamente la proprietà Text e la proprietà ErrorMessage dei controlli di input, ed in tal modo, al verificarsi di un errore, ed in corrispondenza del controllo Validator relativo ad un dato campo, verrà visualizzato il contenuto di Text, mentre il contenuto di ErrorMessage sarà visualizzato come elemento della lista del ValidationSummary.
Un secondo scenario possibile è quello di non visualizzare gli errori in corrispondenza dei controlli, impostando la proprietà Display a None, e utilizzare poi un ValidationSummary per dare un riepilogo a fondo pagina.
La figura seguente mostra la pagina web di esempio con una serie di errore di compilazione, mostrati sia con singoli controlli Validator, sia con un ValidationSummary, e anche con una MessageBox.

Fig. 4: Riepilogo degli errori di validazione

CONCLUSIONI

I controlli di validazione ASP.NET rendono la vita di uno sviluppatore più facile, non dovrà più preoccuparsi dello sviluppo di codice JavaScript per realizzare le stesse cose, come si faceva in passato, o come lo fa chi sviluppa in ambienti meno potenti.
La cosa più potente è forse proprio la semplicità, di realizzazione e modifica, bastano poche proprietà per creare moduli sicuri e intelligenti.

Antonio Pelleriti

RUBY ON RAILS IL WEB SU ROTAIE

TUTTI PARLANO DI RUBY ON RAILS, IL PLURIPREMIATO FRAMEWORK PER SVILUPPARE WEB APPLICATIONS BASATO SUL LINGUAGGIO RUBY. IL MOTIVO È SEMPLICE: RAILS È MAGICO. ED È FACILE DA USARE. NON CI CREDETE? PROVATELO

Rails è un framework per sviluppare applicazioni web basato sul linguaggio Ruby. Le imitazioni di Rails per altri linguaggi si sprecano, ma molti imparano Ruby apposta per usare l'originale. Il segreto di "RoR" è la sua semplicità: anziché impazzire con complicati file di configurazione XML, basta seguire una serie di convenzioni sui nomi e sulle directory perché tutto funzioni "automagicamente". Il risultato è che si può sviluppare per il web a velocità incredibili.
Per capire qual è il bello di RoR, bisogna provarlo. Installate tutti i pezzi che vi servono (le istruzioni sono nel box Il campo da gioco) e mettiamoci al lavoro.

MONTARE LE IMPALCATURE

Svilupperemo una semplice messageboard. Aprite il prompt dei comandi, andate nella directory dove volete creare l'applicazione (ad esempio la root del disco) e scrivete:

```
rails messageboard
```

Vedete quella fila di messaggi sullo schermo? RoR sta montando l'impalcatura per un'applicazione di nome messageboard. Il risultato è una directory messageboard, e sotto quella una serie di sottodirectory: app per il codice, config per i (pochi) file di configurazione, eccetera. In particolare, script contiene una serie di utilissimi programmini Ruby. Usiamone subito uno. Aprite un altro prompt dei comandi, andate nella directory messageboard e scrivete:

```
ruby script\server
```

Avete appena chiesto all'interprete Ruby di lanciare lo script server, un semplice server web configurato per ascoltare sulla porta 3000. Quindi ogni applicazione Rails include un web server. Perché no? E' il modo ideale per provare il codice mentre lo si scrive. Aprite un browser e andate all'indirizzo http://localhost:3000. Dovrebbe apparire una pagina di presentazione di Ruby on Rails.
Verrebbe voglia di mettere subito le mani nella messageboard. Ma prima conviene fare una breve digressione per capire in che modo Rails mappa le nostre URL sul codice dell'applicazione.

URL NELLA NOTTE

Rails segue il classico pattern Model-View-Controller. L'applicazione è divisa in tre parti: modelli che contengono la logica, "controller" che interpretano i comandi dell'utente e viste che mostrano il risultato. I modelli e i controller sono classi Ruby, e le viste sono HTML con un po' di codice Ruby infilato dentro.
Per collegare i tre componenti del modello, Rails usa una semplice convenzione sui nomi. Se ad esempio apriamo l'indirizzo http://localhost:3000/test/ping, Rails cerca un controllo di nome test_controller.rb nella cartella app/controllers dell'applicazione. Questo file deve contenere una classe Ruby di nome TestController, che deve contenere un metodo di nome prova. Quindi prova è il nome dell' "azione" che viene eseguita. Se non conoscete la programmazione ad oggetti, potete pensare ai metodi come a delle specie di funzioni, e alle classi come a dei contenitori di metodi. Ma la classe non esiste ancora, quindi se proviamo ad aprire la pagina otteniamo un messaggio di errore nel browser:

```
Routing Error
Recognition failed for "/test/ping"
```

Se fossimo già esperti potremmo scrivere da soli il TestController – ma è più facile chiedere a Rails di generarlo per noi. Nella directory messageboard scrivete:

```
ruby script\generate controller Test
```

Lo script generate prende come parametro il tipo di oggetto che si vuole generare (in questo caso un

controller), seguito dal suo nome. Il risultato è un file di nome test_controller.rb nella directory app/controllers. Apritelo, e ci troverete una semplice dichiarazione di classe:

```
class TestController < ApplicationController
end
```

Per chi conosce la programmazione object-oriented (gli altri siano tolleranti fino al prossimo paragrafo): il simbolo '<' è l'operatore di ereditarietà, e dice che TestController eredita dalla classe ApplicationController (una semplice classe generata nel file application.rb), che a sua volta eredita da ActionController::Base – una delle "magiche" classi di Rails.
Cosa succede ora se proviamo ad aprire http://localhost:3000/test/ping? Questa volta va un po' meglio: al posto di un errore di routing, appare:

```
Unknown action
No action responded to ping
```

Quindi Rails ha trovato il controller test, ma non l'azione ping. Rimediamo scrivendo un metodo di nome ping nel controller:

```
class TestController < ApplicationController
  def ping
    render :text => "Sono qui! Mi leggi?"
  end
end
```

Abbiamo usato un metodo di Rails che si chiama render (grazie alla flessibile sintassi di Ruby, questa riga sembra più uno strano linguaggio fatto apposta per la web che una normale chiamata di metodo). La roba che segue il nome render il metodo sono il nome e il valore di un parametro. Questo codice chiama la funzione render, assegnando la stringa al parametro text. Salvate il file, ricaricate la pagina nel browser e godetevi l'equivalente Rails di "Hello, world". Ah, non è necessario riavviare il server: una delle cose più belle di Rails è che i risultati di ogni modifica sono immediati.
Quindi, ricapitolando, ecco il formato delle URL di Rails:

http://server/nome_controller/nome_azione?parametro1=valore1;parametro2=valore2

Se omettete il nome dell'azione, Rails cerca un'azione che si chiama index. E cosa succede ai parametri della chiamata HTTP? Come vedremo tra poco, è possibile leggerli dal codice del controller.
Tutto molto bello. Ma di solito non vogliamo mettere del normale testo nel browser: quello che vogliamo è un po' di sano HTML. Aprite la cartella app/views. Quando abbiamo creato il controller test, Rails ha creato anche una cartella app/views/test per le sue viste. Create un file di nome ping.rhtml con questo contenuto:

```
<h1>Calendario</h1>
<%= Date.today %>
```

Questo è quasi un normale file HTML, ma include dei tag speciali: '<%' e '%>'. Tutto quello che è contenuto tra questi tag è codice Ruby, che il server interpreta ogni volta che deve fare un rendering della vista. Se il codice inizia con un segno di uguale, il risultato del codice viene anche infilato nell'HTML. In questo caso, il valore dell'espressione tra i due tag è semplicemente la data di oggi. Provate a ricaricare la pagina nel browser, e... be', vedrete ancora la scritta che arriva dal controller. Il motivo è che l'azione ping ha già specificato cosa voleva visualizzare chiamando il metodo render, quindi Rails non ha cercato la vista. Ma basta cancellare la chiamata a render (o anche l'intero metodo ping) dal controller, e vedrete apparire la pagina calendario. In questo caso la nostra azione non specifica cosa si debba mostrare nel browser, quindi Rails ha pensato bene di cercare una vista con il nome giusto per questa azione di questo controller. L'ha trovata, ha eseguito il codice che c'era dentro e ha restituito l'HTML risultante al browser.
Ora che sappiamo come Rails mappa le URL del browser sul controller, e come recupera le viste, possiamo cominciare a lavorare sulla messageboard.

IL GRIGIO E TRISTE DATABASE

Aprite SQLyog (o qualsiasi altro front-end per MySQL) e create un database di nome message-

IL CAMPO DA GIOCO

Per usare Ruby on Rails, dovete prima di tutto installare il linguaggio Ruby. Mentre scrivo, la versione stabile più recente è la 1.8.4 (che troverete sul CD). L'installazione sotto Windows include un editor, un semplice IDE e RubyGems, il sistema di packaging che useremo per installare RoR. Per provare se l'installazione ha avuto successo, aprite un prompt dei comandi e scrivete:

```
ruby -v
```

Se appare la versione di Ruby installata, basta un semplice comando per installare Rails:

```
gem install rails –remote
```

Abbiamo appena chiesto a RubyGems di scaricare e installare Rails e tutte le librerie necessarie. Confermate tutte le installazioni. Quando avete finito, potete vedere la lista dei pacchetti installati con il comando gem list. Per scrivere questo articolo abbiamo usato la versione 1.1.6 di Rails.

board_development. Per farlo dovete prima connettervi a MySQL, dandogli nome utente e password. Poi selezionate root@localhost e scegliete Create Database dal menu contestuale. Ora ci serve una tabella per i messaggi della messageboard. Chiamatela messages (i nomi sono importanti – non abbreviateli). La prima cosa da mettere in tutte le tabelle quando si lavora con Rails è un campo id. Createlo come un intero e specificate che è una chiave primaria generata automaticamente dal database. Create anche un campo per il titolo (title), uno per il testo (body) e uno per il nome dell'autore (author). Specificate che title e author sono di tipo varchar. Per il body ci serve più spazio, quindi dategli il tipo text. Aggiungete anche un campo created_on di tipo datetime per conservare la data di creazione del messaggio. Ecco lo schema della tabella messages in SQLyog:

Field Name	Datatype	Len	Default	Collation	PK?	Binary?	Not Null?	Unsigned?	Auto Incr?	Zerofill?
id	int	11			☑	☐	☑	☐	☑	☐
title	varchar	200		latin...	☐	☐	☐	☐	☐	☐
body	text			latin...	☐	☐	☐	☐	☐	☐
author	varchar	200		latin...	☐	☐	☐	☐	☐	☐
created_on	datetime				☐	☐	☐	☐	☐	☐

***Fig. 1:* La tabella che rappresenta la nostra MessageBoard**

Ora che abbiamo il database dobbiamo dire a Rails dove trovarlo e come aprirlo. In realtà, visto che abbiamo rispettato tutte le naming convention, Rails conosce già il nome del database. Dobbiamo solo specificare una password. Andate nella directory messageboard/config e aprite il file database.yml. Come vedete Rails ha già scritto una configurazione di default dove manca solo la password. Cercate la sezione development e scrivete la password nel campo password:

```
development:
  adapter: mysql
  database: cookbook_development
  username: root
  password: lamiapassword
  host: localhost
```

Fine della configurazione. Non è stato difficile, vero? Se avete lasciato vuota la password di MySQL, potete anche ignorare completamente la configurazione. Una volta tanto (e per l'ultima volta) ci tocca riavviare il server perché si connetta al DB.
Ora possiamo scrivere il codice che gestisce i messaggi. Non ci vorrà molto: cinque o sei secondi al massimo.

LA MAGIA DI RAILS

Questa volta non vogliamo generare solo il controller, ma l'intera impalcatura ("scaffolding") per i messaggi: modello, vista e controller.

```
ruby script\generate scaffold Message
```

Prima di guardare i file generati, possiamo subito provare l'applicazione. Aprite l'indirizzo http://localhost:3000/messages. Rails cercherà l'azione index nel controller messages – tutta roba che è stata appena generata. La piccola applicazione messa in piedi da Rails permette già di elencare, inserire, modificare cancellare e visualizzare i messaggi.

```
<immagine: nuovo_messaggio.gif>
```

Come al solito, la roba importante è nella directory app. Andiamo in app/controller, e come ci aspettiamo troveremo un file di nome messages_controller.rb. Questa volta, però, la classe Messages Controller include anche un bel po' di codice:

```
class MessagesController < ApplicationController
  def index
    list
    render :action => 'list'
  end
  ...
```

Oltre all'azione index, Rails ha generato il codice per le altre azioni comuni, come create e destroy. Rails scrive esplicitamente il codice delle azioni per rendere più facile la vita a chi lo vuole studiare o modificare, ma in qualsiasi momento è possibile cancellare tutti questi metodi e sostituirli con un'unica riga di codice ("scaffold: message"). Noi lo terremo, perché ci interessa capire cosa sta facendo Rails dietro le quinte.
L'azione index fa due cose: prima di tutto chiama un'altra azione, di nome list, che elenca tutti i messaggi che trova nel database. Poi il controller chiama render per specificare cosa deve essere mostrato nel browser. Sappiamo che possiamo omettere la chiamata a render se abbiamo una vista con lo stesso nome dell'azione. Ma non esiste una vista index, quindi il metodo usa la vista collegata all'azione list. In conclusione, chiamare index e chiamare list è la stessa cosa (potete chiamare direttamente list aprendo l'indirizzo http://localhost:3000/messages/list). Ecco il codice di list:

```
def list
  @message_pages, @messages = paginate
                                  :messages, :per_page => 10
end
```

Questo codice è un po' difficile da spiegare perché si occupa anche della paginazione: se i messaggi sono più di dieci, li divide su pagine diverse. Lasciando da parte la paginazione, la cosa interessante è che questa riga assegna un valore alla variabile @messages.

Per i programmatori object-oriented: in Ruby il simbolo @ prima di un nome identifica i campi, quindi @messages è un campo di MessagesController. Ruby è un linguaggio dinamico, quindi i campi e le variabili non devono essere dichiarati: semplicemente appaiono la prima volta che gli assegniamo un valore.
A cosa serve @messages, e perché è un campo invece di una normale variabile? Il motivo è che i campi, a differenza delle variabili, continuano ed esistere anche quando si esce dal metodo. Quando il metodo termina, Ruby va a cercare la vista per il rendering – in questo caso, visto che non abbiamo specificato altrimenti, Ruby cercherà una vista di nome list.rhtml. Questa vista avrà probabilmente bisogno di qualche dato che è stato calcolato nel controller. I campi @message_pages e @messages sono appunto usati nella vista, come vedremo tra poco. In generale, il controller crea un campo ogni volta che vuole mostrare qualcosa alla vista.
Azioni come list agiscono su tutti i messaggi. Altri comandi, come edit, hanno bisogno di specificare un particolare messaggio. Per dirgli quale, bisogna passargli l'id del messaggio come parametro. Se scriviamo http://localhost:3000/messages/edit?id =1, editiamo il messaggio con identificativo 1. Ecco il codice di edit:

```
def edit
  @message = Message.find(params[:id])
end
```

Il metodo params ci permette di accedere direttamente ai parametri sulla URL. Scrivendo params[:id] recuperiamo il parametro id. Questo codice non fa che cercare il messaggio con l'id specificato grazie al metodo Message.find. Il risultato viene messo in un campo @message per renderlo visibile alla vista edit.
Proprio come edit, la maggior parte degli altri metodi del controller usano la classe Message del modello. Andiamo a vedere com'è fatta.

UNA CLASSE MODELLO

La classe *Message* è nella directory app/models, in un file di nome message.rb:

```
class Message < ActiveRecord::Base
end
```

Questa classe non include ancora codice, ma la sua superclasse ActiveRecord::Base le mette già a disposizione metodi potenti come find. Meglio ancora, ActiveRecord mappa gli oggetti Ruby sul database: una tabella diventa una classe, ciascun record diventa un oggetto, e ciascun campo diventa un campo dell'oggetto. Quindi questa classe del modello mantiene il collegamento tra i nostri oggetti Message e la tabella messages. Rails usa anche altre convenzioni per trattare i campi in modo intelligente, come il già citato campo id. Un altro esempio è il campo created_on. Non abbiamo scelto questo nome a caso: se un campo si chiama così, Rails lo inizializza automaticamente quando viene creato un nuovo oggetto.
Facciamo un esempio. Create un paio di messaggi attraverso l'applicazione web. Poi lanciate la console di Rails, che vi permette di manipolare gli oggetti in modo interattivo:

```
ruby script\console
```

Ora abbiamo accesso agli oggetti del modello esattamente come se fossimo un controller. Andiamo a pescare il messaggio con id uguale a 1 e mettiamo in una variabile:

```
m = Message.find(5)
```

Ora possiamo accedere ai campi della tabella esattamente come se fossero campi dell'oggetto. Provate ad inserire una dopo l'altra queste righe:

```
m.title
m.body
m.author
```

Quando avete finito, uscite dalla console scrivendo exit.
Le classi del modello non sono utili solo per accedere ai dati. Sono anche anche il contenitore ideale per tutta la logica che riguarda i messaggi. Ad esempio, una delle responsabilità del modello è la validazione dei dati. E' facile mettere in crisi la messageboard lasciando ad esempio qualche campo vuoto. Per impedirlo basta aggiungere un paio di righe alla classe Message:

```
class Message < ActiveRecord::Base
 validates_presence_of :title
```

PER CHI NON USA WINDOWS

Se usate un altro sistema operativo non dovreste avere problemi a seguire questo tutorial. Le uniche differenze sono nei package che dovete installare. In particolare dovete usare la versione per il vostro OS di Ruby (da http://www.ruby-lang.org). Non tutte le installazioni di Ruby includono RubyGems, il sistema di packaging di cui avete bisogno per installare Rails. Una volta che avete Ruby e RubyGems, l'installazione di RoR è identica. Dovete anche installare una versione per il vostro sistema operativo di MySQL (http://dev.mysql.com) eun qualsiasi front-end per MySQL. Se siete dei maghi di SQL potete anche fare a meno del front-end.

```
  validates_presence_of :author
end
```

Ora i campi title e author non possono più essere lasciati in bianco. Per verificarlo, tornate all'applicazione e provate ad inserire un messaggio senza titolo.

```
<immagine: validazione.gif>
```

E ora facciamo visita all'ultimo componente della trinità MVC: la vista.

CAMERA CON VISTA

Abbiamo detto che dopo ogni azione Rails ha bisogno di "qualcosa" per fare il rendering. Se non specifichiamo altrimenti, questo qualcosa è una vista con lo stesso nome dell'azione. Ma come sono organizzate le viste? Aprite la directory app/views, e ci troverete dentro una cartella per ciascun controller. Aprite messages, ed ecco apparire le viste: cinque file con estensione rhtml. Uno di questi, _form.html, è un "parziale" - un pezzo di vista usato dalle altre viste. Gli altri sono normali template rhtml come quello che abbiamo visto qualche paragrafo fa, ma un po' più complicati. Ad esempio, andate a guardare la vista list.rhtml. Questo codice usa alcune funzionalità avanzate, compresa la paginazione, quindi possiamo scriverci da soli un esempio più semplice. Modifichiamo list.rhtml:

```
<h1>L'allegra messageboard</h1>
<% for message in @messages %>
   <%= link_to message.title, :action => 'show', :id
                                          => message %>
   <i> - <%= message.author %>, <%=
          message.created_on.to_s(:long) %></i><br>
<% end %>
<p> <%= link_to 'Nuovo messaggio', :action =>
                                        'new' %> </p>
```

Abbiamo eliminato un po' di roba, come il link all'editing dei messaggi e il supporto alla paginazione. Prima di andare avanti dobbiamo rimuovere la paginazione anche dal metodo list del controller, se no rischiamo di non vedere mai altro che i primi dieci messaggi. Scommetto che riuscite ad intuire da soli cosa fa questo codice in MessagesController:

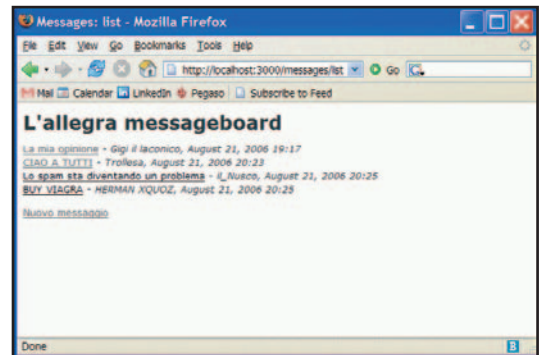



Fig. 2: L'applicazione in esecuzione

```
class MessagesController...
  def list
    @messages = Message.find_all
  end
```

Potete vedere la nuova vista aprendo l'indirizzo http://localhost:3000/messages/list.
Torniamo alla vista:

```
<% for message in @messages %>
  ...
<% end %>
```

Questo codice Ruby "embedded" nell'HTML accede al campo @messages, che è stato inizializzato dal controller e contiene tutti i messaggi. E' un ciclo for che viene chiamato una volta per ciascun oggetto in @messages, e assegna ciascun messaggio a turno alla variabile message. All'interno del ciclo, alcuni campi del messaggio vengono inseriti nell'HTML:

```
<%= link_to message.title, :action => 'show', :id
                                          => message %>
<i> - <%= message.author %>, <%=
          message.created_on.to_s(:long) %></i><br>
```

Abbiamo scritto l'autore così com'è, e abbiamo formattato la data di creazione per renderla più leggibile. Il titolo del messaggio, invece, viene stampato in modo diverso: il comando Rails link_to lo mostra come un link ad un'azione nel controller. L'azione in questo caso è show, che ha bisogno di un identificativo del messaggio. L'identificativo viene infilato nel parametro id, quindi chiamare questa azione dalla vista è equivalente a scrivere l'indirizzo http://localhost:3000/messages/show?id=x, dove x è l'id . Notate che basta scrivere :id => message anziché :id => message.id per far capire a Rails quello che vogliamo fare.
Se volete divertirvi un po', potete provare a modificare le altre viste. Quando avete finito, possiamo tornare a lavorare sul modello.

INDIRIZZI BEN FATTI

Ruby ci tiene molto ad avere delle URL piacevoli e comprensibili dagli esseri umani. Un esempio è la scorciatoia usata per il comunissimo parametro id: anziché scrivere http://localhost:3000/messages/edit?id=1, posso usare il più leggibile indirizzo http://localhost:3000/messages/edit/1.

RELAZIONI PERICOLOSE

Una messageboard serve a poco se non si può rispondere ai messaggi. Per semplificare le cose, costruiamo le risposte ai messaggi come una tabella a parte (in una vera messageboard, le risposte ai messaggi sono a loro volta messaggi – diciamo che la nostra è una via di mezzo tra una messageboard e un blog). Aprite il database, e create una tabella replies. Datele dei campi id, body e author identici a quelli della tabella messages. Aggiungete anche un campo intero message_id (dategli la stessa lunghezza che state usando per i campi id). Questo campo è una chiave esterna che dovrà contenere l'id del messaggio a cui la risposta fa riferimento.

Grazie alle solite convenzioni, Rails "capisce" a cosa serve il campo message_id: se un campo si chiama name_id, il framework si aspetta che contenga l'identificativo di un record nella tabella names. Ora abbiamo la struttura nel database per stabilire una relazione uno-a-molti tra risposte e messaggi: una risposta fa riferimento ad un messaggio, il che implica che ciascun messaggio può avere molte risposte. Abbandoniamo per sempre il database e torniamo al codice.

Questa volta non abbiamo bisogno di creare viste e controller, perché non vogliamo manipolare le risposte attraverso un'interfaccia predefinita come abbiamo fatto per i messaggi – abbiamo piani più ambiziosi. Dobbiamo generare solo il modello. Il singolare di "replies" è "reply", e Rails conosce bene i plurali inglesi:

```
ruby script\generate model Reply
```

Ora abbiamo una classe reply.rb nella directory app/model. Se vogliamo possiamo aggiungere un po' di validazioni per evitare che qualcuno demolisca la nostra applicazione lasciando un campo in bianco:

```
class Reply < ActiveRecord::Base
  validates_presence_of :body
  validates_presence_of :author
end
```

Ora dobbiamo spiegare a Rails come funziona la relazione tra messaggi e risposte. In parte, il framework lo "intuisce" già dalla presenza del campo message_id nella tabella replies. Ma per completare l'opera, dobbiamo confermare che le due tabelle sono effettivamente legate: una reply appartiene a un messaggio, e un messaggio ha molte replies. Per farlo basta aggiungere un paio di righe alle due classi del modello:

```
class Reply...
  belongs_to :message

class Message...
  has_many :replies
```

Fatto. Ora tutti gli oggetti Reply hanno un campo message che contiene il loro Message, e ciascun Message ha un campo replies che contiene la sua lista di Reply. Apriamo la vista show.rhtml dei messaggi e cambiamola in modo che stampi, oltre al messaggio, tutte le risposte:

Field Name	Datatype	Len	Default	Collation	PK?	Binary?	Not Null?	Unsigned?	Auto Incr?	Zerofill?
id	int	11			☑	☐	☑	☑	☑	☐
body	text			latin...	☐	☐	☐	☐	☐	☐
author	varchar	200		latin...	☐	☐	☐	☐	☐	☐
created_on	datetime				☐	☐	☐	☐	☐	☐
message_id	int	11			☐	☐	☐	☐	☐	☐
					☐	☐	☐	☐	☐	☐

Fig. 3: ***La tabella modificata***

```
<h1><%=h @message.title %></h1>
<p><%= @message.body %></p>
<i><%= @message.author %>, <%=
      @message.created_on.to_s(:long) %></i><hr>
<% for reply in @message.replies %>
  <p><%= reply.body %></p>
  <ul><i><%= reply.author %>, <%=
         reply.created_on.to_s(:long) %></i></ul>
<% end %>

<p><%= link_to 'Rispondi', :action => 'reply', :id
                              => @message %> |
<%= link_to 'Tutti i messaggi', :action => 'list'
                                          %></p>
```

Basta cliccare sul titolo di un messaggio nella vista list per aprire l'azione show. Visto che c'eravamo abbiamo anche modificato la formattazione dei dati. Per poter vedere le risposte dobbiamo poterle inserire, e questo è il lavoro del penultimo link:

```
<%= link_to 'Rispondi', :action => 'reply', :id =>
                              @message %> |
```

Non cliccateci subito: questo link chiama l'azione reply sul controller, che ancora non esiste. Quest'azione avrà a sua volta una vista, che

MODEL, VIEW, SPAGHETTI

I principianti dello sviluppo web tendono a spargere il codice tra controller e viste, con il risultato di trovarsi presto invischiati in un melmoso stagno di codice-spaghetti. E' meglio se i controller contengono solo il codice di alto livello per manipolare gli oggetti e le viste solo roba piccola come la formattazione dei campi e qualche semplice ciclo. Sarà il modello ad occuparsi della vera e propria logica dell'applicazione.

servirà a inserire la risposta. E più facile vederlo che spiegarlo, quindi facciamo tutto d'un fiato, cominciando con il metodo reply del controller:

```
class MessagesController...
  def reply
    @message = Message.find(params[:id])
  end
```

Questo metodo recupera il messaggio a cui vogliamo rispondere e lo mette nel campo @message perché sia accessibile alla vista reply. Ora ci serve la vista. Create il file reply.rhtml nella directory app/views/messages e copiateci dentro questo codice:

```
<%= start_form_tag :action => 'post_reply',
                      :to_message => @message.id %>
  <p><%= text_area 'reply', 'body'  %></p>
  <p><label for="reply_author">Da:
                                        </label><%=
                     text_field 'reply', 'author'  %></p>
  <%= submit_tag "Rispondi" %>
<%= end_form_tag %>
<%= link_to 'Annulla', :action => 'show', :id =>
                                         @message %>
```

Qui le cose si complicano un po', perché il gioco di squadra tra viste e controller diventa più articolato. Ricapitoliamo: l'utente ha cliccato sul link “Rispondi” nella vista show, e ha quindi invocato il metodo reply nel controller. Il metodo ha recuperato il messaggio a cui rispondere per passarlo alla vista reply.html. Quest'ultima usa il metodo di Rails start_form_tag per mostrare una form all'utente. Quando l'utente compila e spedisce la form, il metodo start_form_tag chiama a sua volta un'altra azione dal controller, post_reply, passandole un parametro to_message dove infila direttamente l'id del messaggio a cui rispondere (un'informazione che è arrivata fin qui sin dalla vista show). Ricordatevi che i controller comunicano con le viste creando dei campi, e le viste comunicano con i controller usando dei parametri quando chiamano le azioni. Ora dobbiamo scrivere post_reply:

```
class MessagesController...
  def post_reply
    @message =
                  Message.find(params[:to_message])
    reply =
              @message.replies.create(params[:reply])
    if reply.valid?
      render :action => 'show'
    else
      render :action => 'reply'
    end
  end
```

Questa azione ha appena ricevuto un parametro che si chiama to_message, e che contiene l'id di un messaggio. Quindi, per prima cosa recupera questo messaggio. Poi crea una nuova risposta e la mette nella variabile reply. Infine si chiede se la risposta è valida (se ad esempio l' autore vuoto, la risposta non è valida). Se la risposta è valida, viene mostrata la vista show.rhtml. Se non lo è, restiamo fermi sulla vista reply.rhtml.
Perché tutto questo lungo giro? Non avremmo potuto infilare la form per le risposte direttamente nella vista show.rhtml? Be' sì. Abbiamo complicato le cose quel tanto che basta per mostrare come il flusso di un'applicazione Rails passa continuamente tra controller, modello e viste. E a proposito di modello...

CONCLUSIONI

Nella directory messageboard troverete un file di Rake, l'equivalente Ruby di programmi come make o Ant. Questo file permette di fare tante cose utili, incluso calcolare le metriche dell'applicazione. Sulla riga di comando, scrivete:

```
rake stats
```

La mia versione della messageboard contiene 80 righe di codice, escluse le viste e incluse tutte le azioni generate da Rails nel controller. Ho provato a sostituire le azioni che non abbiamo toccato con la riga “scaffold: message”.
Il risultato sono 39 righe di codice! Quando i fan dicono che Rails permette di fare cose incredibili con pochissimo codice, non scherzano. La nostra messageboard ha ancora parecchia strada da fare prima del suo debutto su Internet.
Rails offre funzionalità di security, migrazione dei dati e supporto per i web service, tutto nel suo inconfondibile stil, AJAX e molto altro ancora. Qualcuno ha detto che un buon frameworkrende facili le cose facili, e possibili quelle difficili. Questa definizione sembra fatta apposta per Ruby on Rails. Anche se non è un giocattolo, è facile dimenticarsene e giocarci fino all'alba...

Paolo Perrotta

ZEND FRAMEWORK UN NUOVO MONDO

CLASSI APPOSITAMENTE STUDIATE PER ABBASSARE I TEMPI DI SVILUPPO. PHP È PRONTO PER PASSARE DALLA PROGRAMMAZIONE TRAMITE COSTRUTTI DI BASE AD UNA LOGICA MAGGIORMENTE ORIENTATA AGLI OGGETTI E AI PATTERN PIÙ UTILI. VEDIAMO COME

C'era una volta PHP. Ovvero un linguaggio procedurale con una scarsa attitudine alla programmazione ad oggetti. Nonostante questo, grazie alla sua flessibilità e all'enorme disponibilità di costrutti di base e alla sua leggerezza si conquistò la posizione di leader della rete. Venne poi PHP5, quasi identico al precedente ma decisamente più orientato agli oggetti. In attesa di PHP 6 è adesso tempo di framework!

COSA SI INTENDE PER PHP FRAMEWORK?

La risposta è semplice, un'insieme di classi omogenee per sintassi e logica di programmazione che mettono a disposizione del programmatore una serie di oggetti avanzati costruiti attorno ai costrutti di base. L'esempio è semplice. Prendiamo la generazione di un PDF, volendo partire dalle basi con PHP dovremmo costruire tutte le funzioni atte a generare i report. Avendo a disposizione un framework, probabilmente avremmo a disposizione una classe *gestiscipdf* con un metodo **generareport** a cui passare qualche parametro e il gioco sarebbe fatto. In questo numero di ioProgrammo analizzeremo un nuovo Framework di cui Zend ha reso da poco disponibile una preview release. La versione attualmente disponibile è la preview 0.1.5. Il sito di riferimento per scaricare e installare il necessario è http://framework.zend.com

GLI INGREDIENTI

Una volta eseguita l'installazione, avremo a disposizione diverse classi, ciascuna delle quali inserita in un file con il nome che la rappresenta. In particolare

- Cache
- Config
- Db
- Exception
- Feed
- Filter
- Json
- Log
- Mail
- Mime
- Pdf
- Uri
- View

Inizieremo con il fare qualche esempio con la classe Feed che per la sua semplicità si presta bene a far comprendere qual è la logica dello Zend Framework.
Diamo un'occhiata al seguente script:

```
<?
require_once("Zend/Feed.php");
$feed = new Zend_Feed();
$myfeed=$feed-
        >import('http://news.google.com/?output=rss');
foreach ($myfeed->items as $item) {
        print '<a href="'.$item->link().'">'.$item-
                >title().'</a><br>';
}
?>
```

COME INIZIARE

È sufficiente scaricare lo Zend Framework all'indirizzo http://framework.zend.com e decomprimerlo in una directory dell'hard disk raggiungibile da una direttiva include di PHP. Se intendete utilizzare il framework molto spesso, potete anche centralizzarlo includendo il suo path nella direttiva include del php.ini come segue:

```
include_path =
.:/usr/share/php:/usr/share/php/ZendFramework-0.1.4/library
```

Ovviamente il path qui presente è rappresentativo della situazione del mio hard disk, dovrete adattarlo alla vostra situazione. Per gli utenti Windows, ricordatevi che il path deve essere scritto seguendo la sintassi:

```
c:\php\includes
```

Nella prima riga viene incluso il file Feed.php che contiene le classi del framework dedicate alla gestione dei feed RSS. Viene poi creato un oggetto **$feed** di classe **Zend_Feed**. L'oggetto in questione espone un metodo **import** che prende come parametro l'indirizzo del feed rss e restituisce a sua volta un oggetto che può essere iterato attraverso la sua proprietà **items**. È esattamente quello che abbiamo fatto noi nel ciclo di **foreach** contenuto nelle ultime tre righe del codice. In tutto si tratta di sei righe di codice! Un bel passo avanti se si pensa che senza utilizzare il framework, avremmo dovuto creare una classe apposita per gestire il parser del feed RSS e per prelevarne il contenuto.

IL MODEL VIEW CONTROLLER

Ok, non ne potete più del MVC. Ogni volta che qualcuno vuol fare qualcosa, viene tirato in ballo questo pattern! Come tutti ormai saprete si tratta di un modo di programmare che divide il codice in tre parti ben separate

Il modello
La vista
Il controller

Il modello contiene le interfacce, le classi e tutto ciò che concorre alla creazione della logica di business; la vista contiene le strutture necessarie a produrre l'output verso l'utente; il controller gestisce il flusso d'esecuzione del programma. Ora, per quanto voi possiate detestare questo pattern, se volete creare software professionali facilmente manutenibili e ben strutturati dovrete per forza usarlo! Per cui vediamo come lo Zend Framework implementa questo pattern.
Partiremo dalla gestione del controller. Tanto per chiarirci le idee su cosa vogliamo fare facciamo un esempio pratico. Abbiamo detto che un controller è qualcosa che definisce il flusso d'esecuzione del programma. In poche parole un controller analizza le richieste di un utente e comunica al programma quale funzione scegliere per soddisfarla. Supponiamo che un nostro utente voglia visualizzare l'elenco dei comuni italiani che iniziano per la lettera "A". La sua richiesta potrebbe essere qualcosa del genere

http://www.sitoesempio.it/comuni/elenco/q/A

per lo Zend Framework, come vedremo meglio fra poco, tutte le richieste devono essere gestite da un unico file, l'index.php. Per cui anche questa richiesta deve essere in qualche modo scomposta e passata al file index.php. Per fare questo lavoro dovremo inserire nella document_root dell'applicazione un file .htaccess composto come segue:

```
RewriteEngine On
RewriteRule !\.(js|ico|gif|jpg|png|css|php)$
                                        index.php
```

Grazie a questa regola Apache trasformerà la richiesta appena passata in una serie di POST comprensibili da index.php.
In particolare quando index.php riceverà la richiesta la scomporrà come segue:

```
Controller: comuni
Action: elenco
Key: q
Value: a
```

Questo vuol dire che questa richiesta deve essere soddisfatta da un controller che si chiama "Comuni", al cui interno è definita l'Azione "elenco" alla quale viene passato il parametro "q" il cui valore è "a". All'interno dell'azione "elenco" dovranno essere eseguite tutte le operazioni atte a soddisfare la richiesta.

DALLA TEORIA ALLA PRATICA

Vediamo come implementare con lo Zend Framework quanto detto fin qui.
Il primo passo sarà creare un file index.php al quale verranno rivolte tutte le richieste per il controllo del flusso d'esecuzione del programma. Al suo interno inseriremo le seguenti linee:

```
<?
require_once('Zend/Controller/Front.php');
Zend_Controller_Front::run
                        ('../application/controllers')
?>
```

Il file index.php sarà per noi un file di "bootstrap". Non conterrà la logica di gestione del flusso dei comandi, ma instanzierà la classe **Zend_Controller_Front** e ne richiamerà il metodo **run**. Questo metodo non farà altro che ricercare il controller adatto a gestire la richiesta all'interno del percorso segnalato come parametro al metodo. Prima di procedere è dunque il caso di creare una struttura per la nostra applicazione. Quella consigliata dai creatori del Framework è la seguente:

```
/application
!___/models
!___/views
!___/controllers
/document_root
!___/images
!___/styles
!___index.php
/library
!___/zend
```

Ovviamente all'interno della document_root, va inserito il file .htaccess compilato con le informazioni di cui abbiamo già detto.
Notate che le directory application e library possono anche non essere accessibili direttamente da web. La document_root dovrà invece essere accessibile da web e se siete furbi la farete coincidere con la root della vostra applicazione, così che richiamando http://esempio.it/ si vada fisicamente a richiamare il file index.php contenuto nella directory document_root.
All'interno della directory controllers vanno inseriti tutti i file contenenti le classi che conterranno le Action per le varie richieste. Di default dovremo sempre crearne uno chiamato IndexController.php che verrà richiamato quando nessun altro controller è stato richiesto. Ad esempio se l'utente richiamasse semplicemente

http://www.esempiosito.it/

il file index.php passerebbe automaticamente il controllo al file IndexController.php
nel caso invece di

http://www.sitoesempio.it/comuni/elenco/q/A

il controllo verrebbe passato al file ComuniController.php
per default i nomi dei file contenuti nella directory controller devono essere composti dal nome della richiesta più l'estensione "Controller".

COMPOSIZIONE DEI FILE DI CONTROLLO

Siamo dunque giunti a definire il contenuto dei file di controllo. Abbiamo già detto che al loro interno devono essere contenute le istruzioni per la gestione delle Action. Diamo uno sguardo al file IndexController.php

```
<?
Zend::loadClass('Zend_Controller_Action');
class IndexController extends Zend_Controller_Action
{
            public  function indexAction()
            {
                        echo "Benvenuti nella home
                                                page";
            }
            public function noRouteAction(){
             echo "noroute";
            }
}
?>
```

E al file ComuniController.php

```
<?
Zend::loadClass('Zend_Controller_Action');
class ComuniController extends
                        Zend_Controller_Action
{
            public  function indexAction()
            {
                        echo "Benvenuti nella pagina
                                        dei Comuni";
            }
                        public  function vistaAction()
            {
                        echo "Qui metterò le funzioni
                              per la visualizzare i comuni";
            }
                        public function
                                        noRouteAction(){
             echo "noroute";
            }
}
?>
```

All'interno di questi file vengono definite le classi di controllo. Il loro nome si deve comporre con il nome del file di controllo richiesto più l'estensione controller. Devono inoltre estendere la classe astratta **ZendControllerAction** e pertanto devono implementare almeno i metodi **indexAction** e **noRoute Action**. Il primo verrà richiamato se nessuna funzione sarà richiamata dall'utente, il secondo se non è possibile instradare una richiesta. Ad esempio nel caso

http://www.sitoesempio.it/comuni/vista/q/A

verrà richiamato il controller **comuni** contenuto nel file ComuniController.php, l'azione "**vista**" che sarà gestita dal metodo vistaAction() al quale sarà passato il parametro "Q" il cui valore è "A". Può apparire complesso ad un primo impatto, ma è semplicemente un metodo piramidale. Se avrete la costanza di fare qualche prova pratica vi accorgerete che è tutto molto semplice.

LA VISTA

Se avete ben chiaro il pattern MVC vi accorgete che qui qualcosa non funziona. Noterete che nel controller abbiamo messo delle funzioni di output. In particolare i vari "echo" che servono a visualizzare le stringhe. Vediamo come lo Zend Framework risolve questo problema. Modifichiamo il file Comuni Controller.php come segue:

```
<?
Zend::loadClass('Zend_Controller_Action');
Zend::loadClass('Zend_View');
class ComuniController extends
                                        Zend_Controller_Action
{
            public  function indexAction()
            {
                        $view = new Zend_View();
$view->setScriptPath('../application/views');
                        echo $view-
                                    >render('welcome.php');
            }
            public  function vistaAction()
            {
                        $view = new Zend_View();
                        $view-
                        >setScriptPath('../application/views');
                        echo $view-
                                >render('elencocomuni.php');
            }

            public function noRouteAction(){
             echo "noroute";
            }
}
?>
```

Notate che questa volta non c'è alcuna funzione di output. Piuttosto quando viene invocata ad esempio l'action "vista", viene creato un nuovo oggetto di classe Zend_View. Sarà questo oggetto, tramite il metodo render a produrre l'output. Il contenuto dell'output non sarà però contenuto nel controller ma nel file indicato come parametro al metodo render. Nel nostro caso elencocomuni.php conterrà quanto segue

```
<html>
            <body>
                        <p>qui deve comparire
                                    l'elenco dei comuni</p>
            </body>
</html>
```

IL MODELLO

Resta da risolvere il problema di come gestire la parte di logica dell'applicazione, che come abbiamo già detto deve essere separata sia dal controller sia dalla view. Fino ad ora non abbiamo fatto altro che lavorare con stringhe statiche. Ma dovremo pur cominciare a manovrare qualche dato! Questo non è davvero un problema. Creiamo un file che contenga il nostro modello e mettiamolo nella directory models. Ad esempio il nostro modello potrebbe essere il seguente:

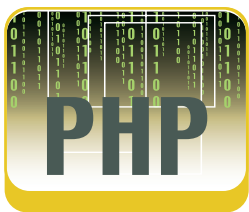
```
<?
class comuniModels {
            public function get_comuni() {
                        $data=
                                    array(
                                                array(
                        'nomecomune'=>'Roma',
                                                'prefisso'=>'06'
                                    )
                        );
                        return $data;
            }
}
?>
```

È davvero banale, ma fa quel che deve fare, ovvero restituire un array associativo con un elenco di comuni. Modifichiamo il controller:

```
                        public  function vistaAction()
            {
                        $view = new Zend_View();
                        $view-
                        >setScriptPath('../application/views');
                        $comuni=new comuniModels();
                        $view->comuni=$comuni-
                                                >get_comuni();
        echo $view->render('elencocomuni.php');

            }
```

Ed infine modifichiamo la view

```
<html>
            <body>
                        <p>
                        <?
                        foreach ($this->comuni as
                                                $key=>$val){
                                    echo $this-
                >escape($val['nomecomune'])."<br>";
                                    echo $this-
                        >escape($val['prefisso'])."<br>";
                        }
                        ?>
                        </p>
            </body>
</html>
```

Praticamente tutto il cuore del sistema sta nel passaggio dei parametri dal controller alla view. In pratica la linea che recita:

```
$view-
>comuni=comuniModels::get_comuni();
```

Abbastanza semplice non trovate? In realtà esistono svariati altri metodi di gestire le view. Ad esempio i template, ma in questo primo articolo introduttivo preferiamo fermarci qui.

LA GESTIONE DEI DATABASE

Fin qui abbiamo usato elementi statici per la nostra applicazione. Il prossimo passo sarà interfacciarla ad un database. Prima di tutto creiamo un database mysql di nome “ioprogrammo_db” e inseriamo al suo interno la seguente tabella

```
CREATE TABLE `comuni` (
`ID_COMUNE` BIGINT NOT NULL
        AUTO_INCREMENT PRIMARY KEY ,
`NOMECOMUNE` VARCHAR( 255 ) NOT NULL ,
`PREFISSO` VARCHAR( 255 ) NOT NULL
) ENGINE = MYISAM CHARACTER SET latin1
        COLLATE latin1_swedish_ci;
```

Passiamo poi a modificare il model che avevamo precedentemente creato come segue:

```
<?
require_once('Zend/Db.php');
class comuniModels {
        private $db;
        function __construct() {
                $params=array(
                        'host'=>'localhost',
                        'username'=>'root',
                        'password'=>'*****',
                        'dbname'=>'ioprogrammo_db'
                        );
                        $this-
>db=Zend_Db::factory('pdoMysql',$params);
        }
                public function get_comuni() {
                $select=$this->db->select();
                $select->from('comuni','*');
                $sql=$select->__toString();
                $rows=$this->db-
                        >fetchAll($select);
                return $rows;
        }
}
?>
```

Cerchiamo di capire quello che succede. Viene incluso il riferimento alla classe Db dello Zend Framework. In questa classe sono contenute tutte le strutture per la gestione dei database.
Dichiariamo una variabile privata $db che utilizzeremo come handle verso il database.
Nel costruttore della classe dichiariamo un array $params contenente tutte le informazioni per la connessione al database.
Inizializziamo l’handle richiamando il metodo Zend_Db::factory passandogli come parametro, il driver che vogliamo utilizzare per la connessione e i parametri di connessione contenuti nell’array. In questo modo ogni volta che verrà creato un nuovo oggetto di classe comuneModels verrà inizializzato il costruttore che metterà a disposizione della classe un handle correttamente connesso verso il database.
Modifichiamo poi il metodo get_comuni. Prima di tutto creiamo un handle al metodo select, poi riempiamolo con la query che vogliamo eseguire.
Invochiamo poi il metodo __toString() per convertire la nostra query in un linguaggio sql comprensibile al db, ed infine eseguiamola con il metodo fetchAll.
I vantaggi di questa soluzione sono immensi. Non abbiamo dovuto eseguire infatti nessuna modifica nè al controller nè alla vista, semplicemente variando il modello possiamo variare a nostro piacimento il comportamento dell’intera applicazione. Se volessimo da domani usare Oracle piuttosto che SQL Server, dovremmo semplicemente variare la stringa di connessione all’interno del modello. Allo stesso modo se volessimo modificare l’aspetto estetico dovremmo semplicemente modificare la vista. Il pattern MVC si rivela in tutta la sua potenza

DEBOLEZZE DEL FRAMEWORK

Nelle nostre sperimentazioni non abbiamo sperimentato alcuna falla nel sistema. L’unica debolezza potrebbe essere rappresentata dall’uso del modulo “rewriterule”. In effetti sotto certe condizioni, trovare una configurazione accurata per questo modulo potrebbe rappresentare un problema. In realtà si tratta di un modulo di Apache e non dello Zend Framework. Tuttavia poiché il corretto funzionamento del framework è legato all’utilizzo di questo modulo, ne deriva anche che difficilmente lo si possa usare con IIS per esempio. In Zend ci fanno sapere comunque che stanno riscrivendo la parte legata proprio alle “route”, ovvero la gestione della scomposizione degli URL e il consequenziale passaggio dei parametri ai vari controller. Quanto questa riscrittura renderà indipendente il framework dal modulo rewriterule non è dato saperlo.

USO DEI PARAMETRI

Fin qui abbiamo eseguito la Select sull'intero database, ma il nostro punto di partenza era:
http://www.sitoesempio.it/comuni/elenco/q/A

dove 'q' era un parametro passato al controller per ottenere l'elenco di tutti i comuni che iniziavano con la lettera 'A'. Ma niente abbiamo detto su come il controller riceve questi parametri e come è poi possibile gestirli dal model. Facciamo qualche esempio. Questa volta siamo costretti a modificare il controller come segue:

```
public  function vistaAction()
        {
                $param=$this->_getParam('q');
                $view = new Zend_View();
                $view-
                >setScriptPath('../application/views');
                $comuni=new comuniModels();
                $view->comuni=$comuni-
                        >get_comuni($param);
    echo $view->render('elencocomuni.php');
        }
```

Notate che abbiamo inserito la riga

```
$param=$this->_getParam('q');
```

infatti i parametri vengono passati al controller all'interno di un array i cui valori possono essere recuperati con il metodo _getParam(nomedelparametro). Per ottenere tutti i valori passati come parametro si può anche usare il metodo _getAllParams(). A questo punto possiamo modificare la riga che richiama il modello e passargli il parametro appena ottenuto. Lo facciamo come segue:

```
                $view->comuni=$comuni-
                        >get_comuni($param);
```

Non ci resta che modificare il modello. Un esempio è il seguente:

```
public function get_comuni($params) {
                $params='%'.$params.'%';
                $select=$this->db->select();
                $select->from('comuni','*')
                        ->
                where('nomecomune like ?',$params);
                $sql=$select->__toString();
                $rows=$this->db-
                        >fetchAll($sql);
                return $rows;
        }
```

Non è decisamente un ottimo esempio di query parametrica, ma è sufficiente a farne capire il funzionamento. Come vedete il metodo ha semplicemente ricevuto il parametro. L'unica nota di interesse è relativa alla linea

```
-> where('nomecomune like ?',$params);
```

dove la condizione viene creata attraverso una query parametrica. La sintassi comunque è esplicativa.

INSERIMENTO DEI DATI

Ormai abbiamo imparato molto sullo Zend Framework. Ipotizziamo di voler fornire il seguente link per l'inserimento di un dato

http://172.16.0.241/comuni/insertnew

Perché tutto funzioni a dovere, dovremo semplicemente creare l'action **insertnewAction** all'interno del controller comuni. Facciamolo come segue:

```
public function insertnewAction() {
                $view = new Zend_View();
                $view-
                >setScriptPath('../application/views');
                echo $view-
                >render('inseriscinuovocomune.php');
        }
```

Al solito non abbiamo fatto altro che instanziare una vista che ci dovrà fornire la maschera per l'inserimento di un nuovo comune. Dovremo ora creare la maschera nel file "inseriscinuovocomune.php" contenuto nella directory "../application/views". Il file in questione conterrà qualcosa del genere.

```
<html>
<head>
</head>
<body>
<p><strong>Inserisci un nuovo
                        comune</strong></p>
<form name="form1" method="post"
                        action="/comuni/create">
  <p>Nome Comune:
    <input name="nomecomune" type="text"
                        id="nomecomune">
  </p>
  <p>
    Prefisso:
    <input name="prefisso" type="text"
                        id="prefisso">
  </p>
    <input name="Salva" type="submit" id="salva"
 value="Salva">
  </p>
</form>
```

```
</body>
</html>
```

Notate che la action è “/comuni/create”. Di conseguenza dovremo implementare questa action nel file comuniController.php.
L’implementazione è la seguente:

```
        public function createAction() {
                $comuni=new comuniModels();
                $param = array(
                        nomecomune =>
                        $_POST['nomecomune'],
                        prefisso =>
                                $_POST['prefisso']
                );
                $view = new Zend_View();
            $view->setScriptPat0
                        ('../application/views');
              $view->esito=$comuni->put_comune
                                    ($param);
              $view->render('esitotransazione.php');
        }
```

Anche in questo caso è abbastanza semplice comprendere la logica di questo spezzone di codice. Nel momento in cui l’utente clicca sul bottone “submit” della form, viene riempito l’array $_POST che sopravvive a livello globale dell’applicazione.
La Action che abbiamo appena implementato non fa altro che prendere i valori di $_POST, valorizzare un array e richiamare un metodo put_comune() che avremo definito nella classe comuniModel.
Infine l’esito della transazione verrà visualizzato in una vista implementata in “esitotransazione.php”.
Non ci resta che definire il metodo in put_comune(). Una possibile implementazione è la seguente:

```
        public function put_comune($params) {
                $table='comuni';
                return $this->db
                        ->insert($table,$params);
        }
```

Inserire un nuovo dato all’interno del database si riduce a due righe di codice! Veramente un gioco da ragazzi! L’ultimo passo sarà definire il contenuto di ‘esitotransazione.php’. Anche in questo caso è facilmente intuibile come la soluzione possa essere semplice:

```
<html>
<body>
<? if ($esito==1) {
        echo "esito positivo";
} else {
        echo "esito negativo";
}

</body>

</html>
```

Di una semplicità sconcertante!

CONCLUSIONI

Lo Zend Framework è veramente enorme. Al suo interno ci sono classi per gestire veramente di tutto, dalla cache alla configurazione, alle session. In questo primo appuntamento abbiamo scelto di presentarvi lo strumento utilizzando come esempio l’implementazione del pattern MVC. Così facendo speriamo di avervi fornito uno sguardo d’insieme al framework.
Con questo non abbiamo assolutamente voluto esaurire la descrizione delle varie funzionalità, nè produrre uno script ottimizzato e affidabile, abbiamo semplicemente affrontato un percorso didattico per potervi illustrare le caratteristiche di questo nuovo strumento.
In futuro affronteremo le varie classi sempre più in profondità, mostrando come il vostro lavoro possa diventare eccezionalmente produttivo tramite l’uso dello Zend Framework.
A margine di questa trattazione vogliamo far rilevare come in molti si ostinino ad utilizzare PHP come un linguaggio procedurale anziché ad oggetti.
Nulla di male in questo, tuttavia vogliamo stimolarvi ad utilizzare questo meccanismo che per potenzialità, affidabilità, garanzie di manutebilità , rappresenta l’unica strada per ottenere applicazioni professionali.

L’ESTENSIONE PDO

Si tratta di un modulo di PHP che consente di rendere indipendente il codice dal database utilizzato. È composta da un modulo generale denominato PDO e da un suo driver specifico che risponde al nome PDO_nomedeldatabase. Per utilizzare questa estensione così come abbiamo fatto noi, dovete abilitarla nel php.ini, come segue:

```
extension = pdo.so
;pdo.dll nel caso di utenti windows
extension = pdo_mysql.so
```

Per la sua installazione è sufficiente inviare a linea di comando il seguente comando

```
PHP_PDO_SHARED=1 pecl install
                        pdo_mysql
```

Se siete utenti windows potete scaricare la dll direttamente dal sito http://pecl.php.net

MEMBERSHIP, ROLES E PROFILE API CON ASP

ASP.NET 2.0 RENDE PIÙ SEMPLICE LA CREAZIONE DI APPLICAZIONI CHE SIANO PROTETTE DA AUTENTICAZIONE, SUPPORTINO I RUOLI E LA PERSONALIZZAZIONE DELL'INTERFACCIA ATTRAVERSO UN PARADIGMA SEMPLICE DA USARE. VEDIAMO COME FUNZIONA

La protezione e personalizzazione di un'applicazione basata sulla versione 2.0 di ASP.NET è un operazione certamente semplice se paragonata a ciò che è necessario fare con la versione precedente.
È indubbio che queste due caratteristiche siano presenti praticamente all'interno di tutte le applicazioni web, dunque l'idea che sta alla base di **Membership, Roles e Profile API** è di fornire un'infrastruttura adeguata a favorire l'implementazione di soluzioni che abbiano, rispettivamente, sistemi di gestione degli utenti, dei ruoli ad essi associati e del profilo utente di ciascuno di essi.
ASP.NET 2.0 per semplificare l'inserimento di queste funzionalità all'interno delle nostre applicazioni si appoggia ad una tecnica nota come **Provider Model**.
Come il nome stesso suggerisce, è essenzialmente un modello teso a favorire lo sviluppo di applicazioni basate su provider, che consentano cioè un alto grado di personalizzazione attraverso la flessibilità di configurazione.
Dunque, per meglio comprendere quanto queste tre API rendano più immediato lo sviluppo, è necessario partire innanzitutto dal Provider Model stesso, che è probabilmente il motivo principale per cui queste tre API hanno riscosso tanto successo tra gli sviluppatori.

IL PROVIDER MODEL DESIGN PATTERN

Il Provider Model è un design pattern, che è un insieme di linee guida da utilizzare in fase di design dell'applicazione web. Un **pattern**, per completezza, è nient'altro che il tentativo di rendere in forma teorica ciò che ci viene dettato dall'esperienza quotidiana. È un modo attraverso il quale diamo un nome ad una problematica ed alla relativa soluzione, che abbiamo visto che si ripete, con una certa costanza, all'interno delle applicazioni.
Nel caso specifico, possiamo vedere il Provider Model come l'unione di altri pattern già esistenti, di cui quello che spicca maggiormente è lo **Strategy Pattern**, che appartiene alla famiglia dei pattern **GOF** (Gang of Four). Se analizzassimo quest'ultimo, ci renderemmo conto che si basa su un concetto tutto sommato semplice: la creazione di una famiglia di algoritmi, incapsulati e resi intercambiabili tra di loro, richiamati in maniera indipendente dal contesto.
Semplificando, potremmo dire che si basano sull'esistenza di un'implementazione generica, che sarà sfruttata esternamente, ed una concreta, che sarà utilizzata da quella generica, per avere effettivamente l'esecuzione delle operazioni previste per una data funzionalità.
Il vantaggio di questo approccio è che cambiando l'implementazione concreta non è necessario cambiare quella generica, che è poi l'unica utilizzata esternamente, cioè dalle applicazioni. L'implementazione concreta è quello che viene appunto definito il **provider**.
Grazie a questa caratteristica è possibile creare applicazioni che, una volta utilizzata l'implementazione generica, che è chiamata **API**, siano in grado di funzionare praticamente all'infinito, dato che se è necessario cambiare lo storage dei dati, è sufficiente solo specificare un nuovo provider. Quest'ultimo è specificato attraverso un file di configurazione, che nel caso delle applicazioni web è rappresentato dal web.config, e caricato quando vengono richiamate le API attraverso un'operazione di **Dependency Injection**.
Inoltre, qualsiasi sviluppatore può creare dei propri provider e quindi aggiungere con relativa facilità qualsiasi funzionalità non fosse prevista, dato che le API non sono nient'altro, in questo modello, che un ponte verso il provider, che è l'effettivo componente che fornisce le funzionalità effettive.
Il Provider Model in realtà è molto più complesso di così, per cui una trattazione più ampia e teorica si può trovare all'indirizzo http://tags.aspitalia.com/Provider_Model/

AUTENTICAZIONE ED AUTORIZZAZIONE

All'interno di ASP.NET, sin dalla versione 1.0, le funzionalità di autenticazione ed autorizzazione sono implementate attraverso degli **Http Module**, che sono tra i componenti più utilizzati da **HttpRuntime**, cioè dall'infrastruttura che sta alla base del ciclo di vita dell'intera applicazione web.
Un HttpModule non è nient'altro che una classe particolare, che implementa l'interfaccia IHttpModule, che nel suo metodo Init si registra per uno o più eventi dell'applicazione.
Nel caso degli **AuthenticationModule**, cioè di quegli HttpModule specifici per l'autenticazione, l'evento è AuthenticateRequest, scatenato dalla classe **HttpApplication**, che rappresenta l'applicazione nel quale il codice e quindi l'HttpModule sono eseguiti.
Questo evento si verifica ogni volta che ASP.NET cerca di effettuare un'autenticazione e ce ne sono ovviamente di differenti a seconda del tipo di autenticazione utilizzata, nel caso specifico 3: **FormsAuthenticationModule**, **WindowsAuthenticationModule** e **PassportAuthenticationModule**.
Il loro uso è rivolto, rispettivamente, alla protezione attraverso una form, le credenziali di Windows o sfruttando Passport.
All'interno dei siti pubblici, è la prima delle tre ad essere quella utilizzata nella quasi totalità dei casi, perchè l'uso di Passport non è aperto al pubblico e la protezione di Windows non funziona attraverso i firewall.
Nella versione 2.0 le novità della Forms Authentication riguardano soprattutto la presenza di meccanismi che fanno sì che sia possibile sfruttarla anche senza cookie, utilizzando l'URL per salvare il ticket di autenticazione, un po' come fa Amazon.
Queste informazioni vanno specificate, come già detto, all'interno del web.config, in questa maniera:

```
<configuration>
  <system.web>
    <authentication mode="forms">
      <forms loginUrl="login.aspx" />
    </authentication>
  </system.web>
</configuration>
```

Una volta specificate le impostazioni di autenticazione, ASP.NET prevede un meccanismo simile per la fase di autorizzazione, in modo da garantire che l'utente possa accedere ad una certa risorsa. In questo caso gli HttpModule sono chiamati **AutorizationModule**, perché intercettano l'evento **AuthorizeRequest** di HttpApplication, che si verifica appunto quando è necessario consentire l'accesso ad una data risorsa.
ASP.NET ha due AuthorizationModule: **UrlAuthorizationModule** e **FileAuthorization Module**. Il primo si occupa di gestire l'accesso agli URL, laddove il secondo invece le confronta sui file.
Nel caso delle applicazioni web pubbliche è il primo dei due ad essere utilizzato. La configurazione si basa sulla definizione delle politiche di accesso all'interno del web.config, potendo impostare sia le policy relative ad utenti che ai ruoli.
Il carattere "*" rappresenta tutti gli utenti (o ruoli), laddove invece "?" rappresenta l'utente che non ha effettuato il login. Con la chiave deny si nega un ruolo o utente, con allow invece si garantisce l'accesso. La definizione di queste policy può essere fatta anche sulla base di una singola pagina, utilizzando il tag location:

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="users" />
      <deny users="*" />
    </authorization>
  </system.web>

  <location path="miapagina.aspx">
    <system.web>
      <authorization>
        <allow roles="ruolo1;ruolo2" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

Le policy vengono impostate secondo un meccanismo di short-circuiting. In pratica, la prima che dovesse verificarsi fa scattare l'uscita dal

PROVIDER AGGIUNTIVI

In tutti quei casi in cui SQL Server non sia il database da utilizzare, è possibile cambiare il provider in modo tale che rifletta quello di proprio interesse. In giro si trovano diversi provider, al momento ecco quelli disponibili:

- **Access** http://msdn.microsoft.com/vstudio/eula.aspx?id=96713a8e-b8d4-4d6e-bb8f-027e6c8e15d8
- **MySQL** http://www.codeproject.com/aspnet/MySQLMembershipProvider.asp
- **SQLite 3.0** http://www.eggheadcafe.com/articles/20051119.asp
- **Oracle** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdasamppet4.asp

ramo delle opzioni, con conseguente non esecuzione dei successivi controlli. È per questo motivo che non dovrebbe mai essere messa come prima opzione <deny users="*" />, perché ha come effetto quello di non far mai verificare le altre policy. Nell'esempio, giustamente, è l'ultima delle opzioni, così da essere applicata solo se le altre non dovessero garantire risultati.
La protezione basata sul web.config va bene nella quasi totalità dei casi, qualora invece si abbia bisogno di poter controllare maggiormente la definizione degli accessi, ad esempio dopo aver eseguito delle istruzioni particolari, la via più semplice è quella di creare un proprio AuthorizationModule. Questo ci consente di continuare ad utilizzare l'infrastruttura di ASP.NET, controllando attraverso una classe esterna le politiche di accesso, senza bisogno di "annegarle" all'interno della singola pagina. Si tratta di creare un HttpModule, come già spiegato in articoli passati apparsi su questa stessa rivista.

MEMBERSHIP API PER GESTIRE GLI UTENTI

Una volta compreso come ASP.NET 2.0 agisca di fronte ad autenticazione ed autorizzazione, è utile dare uno sguardo alla prima delle tre API oggetto di questo articolo.
Come il nome suggerisce, siamo di fronte ad una serie di funzionalità create per rendere possibile l'implementazione legate alla gestione degli utenti. Le Membership API, proprio perché implementano il Provider Model, possono essere ricondotte a due componenti fondamentali:

- le API, implementate attraverso la classe statica Membership;
- i provider, che derivano dalla classe base astratta **MembershipProvider**.

Ogni metodo di Membership, infatti, non fa nient'altro che richiamare il corrispondente metodo all'interno del provider, dopo averne creato un'istanza in base alle informazioni specificate all'interno del web.config.
All'interno di ASP.NET 2.0 ci sono due provider inclusi e pronti ad essere utilizzati e così come Membership e MembershipProvider, si trovano tutti nel namespace System.Web.Security. Questi due provider sono **SqlMembership Provider** ed **ActiveDirectoryMembership Provider**, che come il nome suggerisce, nient'altro sono se non il provider per SQL Server (dalla 7 alla 2005, anche Express o MSDE) e per Active Directory.
La classe Membership ha diversi metodi che consentono di creare utenti, cancellarli, effettuare login, etc, dunque ben si presta, praticamente, ad essere utilizzata in tutte quelle applicazioni web in cui sia necessario avere funzionalità di questo tipo. D'altra parte Membership, di suo, non fa nient'altro che richiamare il provider, dunque una volta implementata un'applicazione, questa funzionerà senza modifiche anche in caso di cambio di provider, poiché quest'ultimo viene definito semplicemente da web.config e non riguarda per niente né il codice né il markup dell'applicazione.
Per rendere ancora più semplice l'implementazione di queste funzionalità, la versione 2.0 di ASP.NET introduce alcuni nuovi server controls di tipo **security**, che automatizzano al 100% questo genere di operazioni, grazie all'utilizzo di un approccio totalmente dichiarativo.
Questi controls sono CreateUserWizard, Login, ChangePassword e PasswordRecovery e coprono il 90% delle necessità presenti in questo ambito.
CreateUserWizard è il controllo deputato all'inserimento di un wizard nella pagina che raccolga le informazioni per creare un utente (username, e-mail e password) e proceda alla creazione dell'utente, invocando il metodo CreateUser di Membership, che a sua volta richiamerà lo stesso metodo offerto dal provider. Questo controllo è molto potente e si inserisce nella sua forma base con poche righe di codice:

```
<form runat="server">
<asp:CreateUserWizard id="wizard1"
                     runat="server" />
```

COME CREARE CUSTOM PROVIDER

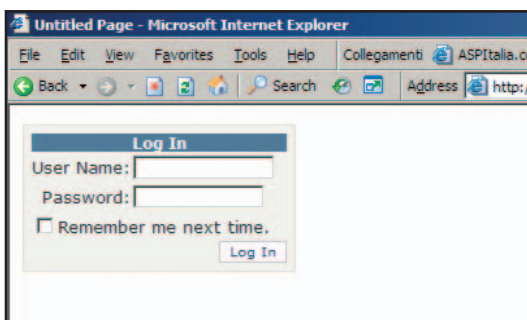
Qualora si abbia la necessità di creare provider personalizzati, la strada più semplice è quella di prendere uno dei provider inclusi in ASP.NET 2.0 e crearne uno derivato, sfruttando l'apposito toolkit disponibile su http://msdn.microsoft.com/asp.net/downloads/providers/default.aspx
Nel caso si abbia bisogno di ulteriori approfondimenti, è possibile leggere questo articolo su http://www.aspitalia.com/articoli/asp.net2/membership_provider.aspx

```
</form>
```

L'effetto con questo control è quello di avere già tutto pronto, senza fare niente e senza dover modificare nient'altro che il provider. Nel caso fosse necessario personalizzare la maschera di iscrizione, questo controllo è un derivato di quello Wizard, dunque può avere sia Step aggiuntivi che una personalizzazione del template. Queste informazioni aggiuntive possono poi essere recuperate intercettando l'evento **CreateUser**, che si verifica quando l'utente è stato correttamente creato attraverso il provider specificato. Il controllo può essere recuperato attraverso un codice del genere:

```
string option = ((DropDownList)
 wizard1.CreateUserStep.ContentTemplateContainer.F
                    indControl("Options")).SelectedValue;
```

In questo esempio viene preso il valore selezionato in una DropDownList il cui ID è "Options", ma ovviamente può essere adattato facilmente a qualsiasi tipologia di controllo e non c'è un limite per il numero degli stessi.
Il controllo Login, invece, inserisce nient'altro che una maschera per fare il login, che in questo caso richiama il metodo ValidateUser della classe Membership. Anche in questo l'inserimento consiste in poche righe di codice:

```
<asp:Login ID="Login1" runat="server" />
```

Attraverso le proprietà che espone, è possibile personalizzarne completamente l'aspetto grafico, così come il testo associato ai vari controlli, per averli anche localizzati in italiano. In questo caso i due eventi di riferimento da intercettare sono LoggedIn e LoginError, che servono rispettivamente ad associare codice all'avvenuto login o ad un errore nello stesso.
ChangePassword e **PasswordRecovery** servono rispettivamente per cambiare la password o recuperare la stessa. Il primo va ovviamente messo su pagina riservati agli utenti autenticati, laddove il secondo deve essere aperto a tutti. Sono facilmente integrabili nelle applicazioni e si trova un esempio nell'allegato a questo articolo.

IL PROVIDER PER SQL SERVER

È vero che le Membership API ed i nuovi controlli lavorano in maniera indipendente, ma è innegabile che nella quasi totalità dei casi debba essere specificato un provider diverso da quello di default, che lavora con **SQL Server 2005 Express**, difficilmente utilizzato in produzione, perché comunque limitato.
Per questo motivo, è utile dare un'occhiata a come funzioni il provider per SQL Server. Per prima cosa, ogni provider ha un tool differente per la sua installazione, che in genere consiste nella creazione di un po' di tabelle all'interno di un database.
Nel caso di quello per SQL Server, Microsoft distribuisce insieme ad ASP.NET un tool a riga di comando, chiamato **aspnet_regsql.exe**, che si trova nella solita directory **%WinDir%\Microsoft.NET\Framework\v2.0.50727**.
Una volta lanciato si occupa di creare nel database specificato tutte le tabelle che servono per Membership, Roles e Profile API. Non tutti i provider vengono distribuiti con tool del genere e questo va comunque sottolineato, perché se si scelgono altri storage, spesso l'operazione di installazione è più macchinosa.
Ad ogni modo, una volta creato il database, va modificato il web.config per fare in modo che contenga il riferimento al provider, in questo modo:

```
<configuration>
  <connectionStrings>
    <add connectionString="Data Source=localhost;
          Initial Catalog=ioprogrammo; Integrated
               Security=SSPI;" name="localhost"/>
```

SUPPORTO PER I PROFILI ANONIMI

Le Profile API supportano anche i profili anonimi. Perché possa funzionare, è necessario per prima cosa aggiungere l'attributo allowAnonymous="true" sulle proprietà definite all'interno del web.config, poi abilitare il supporto per i profili anonimi, sempre da web.config:

```
<anonymousIdentification
   enabled="true" cookieProtection="All"
                                    />
```

Successivamente, sarà possibile creare i profili anche per gli utenti che non abbiano effettuato il login. Nel caso l'utente cambi stato ed effettui l'autenticazione, è possibile intercettare l'evento MigrateAnonymous di Profile, in modo da poter migrare il profilo anonimo all'interno di quello dell'utente.

```
    </connectionStrings>

    <system.web>
      <membership defaultProvider="SqlServer">
        <providers>
          <add connectionStringName="localhost"
               enablePasswordRetrieval="false"
               enablePasswordReset="true"
               requiresQuestionAndAnswer="true"
               applicationName="ioprogrammo"
               requiresUniqueEmail="false"
               passwordFormat="Hashed"
               maxInvalidPasswordAttempts="5"
               minRequiredPasswordLength="7"
               minRequiredNonalphanumericCharacters="1"
               passwordAttemptWindow="10"
               passwordStrengthRegularExpression=""
               name="SqlServer"
    type="System.Web.Security.SqlMembershipProvider,
         System.Web, Version=2.0.0.0, Culture=neutral,
         PublicKeyToken=b03f5f7f11d50a3a" />
        </providers>
      </membership>
    </system.web>
</configuration>
```

Quello che è utile sottolineare sono gli attributi connectionStringName, name e type, che servono rispettivamente a specificare il nome della stringa di connessione, contenuta nell'apposita sezione connectionStrings, il nome del provider (ce ne possono essere anche più di uno specificato) ed il riferimento alla classe che contiene l'implementazione vera e propria. Nel caso di un provider differente, spesso a variare è propria quest'ultima impostazione. L'attributo applicationName, invece, serve per specificare un nome da utilizzare all'interno del database, che così per come è pensato il provider per SQL Server è in grado di supportare più applicazioni contemporaneamente. Per il resto, gli attributi specificati dovrebbero essere di facile comprensione e comunque non vengono quasi mai modificati. In ultima analisi, vale la pena ricordare che questo provider utilizza **stored procedure transazionali**, garantendo un alto grado di performance e consistenza dei dati, e salva la password in formato hashed, con conseguente sicurezza dei dati salvati nel database utenti.

ROLES API, PER GESTIRE I RUOLI

I controlli appena visti sono in grado di funzionare anche in presenza di ruoli. Chi abbia mai provato ad implementare l'autenticazione con ruoli con ASP.NET 1.x sa che non è certo agevole, perché è necessario sovrascrivere il **Principal** generato dalla Forms Authentication con uno proprio. Il Principal non contiene nient'altro che le credenziali dell'utente ed i ruoli a cui appartiene, per cui con la versione 2.0 di ASP.NET questo lavoro di sostituzione viene fatto in automatico attraverso l'HttpModule **RoleManagerModule**, che si occupa di caricare in automatico i ruoli sfruttando le **Roles API**.
Proprio come per le Membership API, anche in questo caso il funzionamento è basato su provider e ne esistono due già pronti, entrambi derivati dalla classe astratta **RoleProvider**. Si tratta di **SqlRoleProvider**, per SQL Server, e **WindowsTokenRoleProvider**, che è invece specifico per l'autenticazione di Windows.
Anche in questo caso va specificato da web.config, in questo modo:

```
<configuration>
  <system.web>
    <roleManager enabled="true"
                 defaultProvider="SqlServer">
      <providers>
        <add connectionStringName="SqlServer"
  applicationName="ioprogrammo" name="SqlServer"

          type="System.Web.Security.SqlRoleProvider,
        System.Web, Version=2.0.0.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </roleManager>
```

Abilitando il supporto per i ruoli, non è necessario fare nient'altro, dato l'HttpModule si occuperà del resto. Si possono creare nuovi ruoli sfruttando il metodo statico CreateRole della classe Roles, così come è possibile aggiungere un utente ad un ruolo attraverso il metodo AddUserToRole. Idealmente queste operazioni vanno fatte attraverso un pannello di controllo, così da rendere possibile la definizione dei ruoli in maniera più pratica. Dove le Roles API danno il meglio è quando cominciamo ad utilizzare il controllo **LoginView**. Si tratta di un controllo che consente di definire tre tipi di template:

UNA CLASSE COME PROFILO

Nel caso in cui si preferisca definire le proprietà del profilo all'interno di una classe, è necessario modificare il web.config perché venga fatto riferimento alla nostra classe:

```
<profile inherits="MyProfile" />
```

La classe creata dovrà ereditare da ProfileBase e per ogni proprietà del profilo sarà necessario prevedere una proprietà specifica nella classe creata.

utente autenticato, anonimo o appartenente ad un ruolo. Il tutto viene fatto in maniera molto semplice, come si può vedere da questo esempio:

```
<asp:LoginView ID="LoginView1" runat="server">
  <RoleGroups>
    <asp:RoleGroup Roles="admin">
      <ContentTemplate>Appartenente al ruolo
                                  admin.</ContentTemplate>
    </asp:RoleGroup>
  </RoleGroups>
<LoggedInTemplate>Autenticato.
                                     </LoggedInTemplate>
<AnonymousTemplate>Anonimo.
                                  </AnonymousTemplate>
</asp:LoginView>
```

A seconda dello stato dell'utente, sarà mostrato il template corrispondente. Si possono annidare diversi controlli di questo tipo, qualora si voglia fare in modo che ci siano template specifici per gruppi di ruoli, perché di default questo controllo è in grado di visualizzare solo uno dei template per volta, benché poi quello con i ruoli ne possa a sua volta avere più di uno, ciascuno per ruolo. Una menzione particolare, infine, ai controlli **LoginStatus** e **LoginName**, che servono rispettivamente a mostrare lo stato del login, associando un link alla pagina di login o logout a seconda dello stesso, ed il nome utente. Sono utili, ma tutto sommato si tratta di controlli che di sicuro sono di contorno rispetto agli altri già analizzati.

GESTIONE DEL PROFILO UTENTE

Per ultimo, è arrivato il momento di dare una rapida occhiata a ciò che le Profile API consentono di fare. Fatta eccezione per il nome utente, qualsiasi altra informazione associata allo stesso deve essere salvata all'interno del profilo, perché non strettamente attinente alle informazioni necessarie a gestire la fase di autenticazione. È questo il motivo per cui praticamente non è necessario creare un provider custom per le Membership API qualora si abbia soltanto la necessità di gestire, ad esempio, Nome e Cognome dello stesso. Per queste cose è più sensato fare uso delle Profile API, che lavorano bene in unione con le Membership API, così come in presenza di profili anonimi. Un profilo è nient'altro che una classe composta da alcune proprietà e la cosa interessante è che il tutto, con ASP.NET, è strongly-typed. C'è la possibilità di definire le proprietà del profilo direttamente da web.config, attraverso il nodo properties del nodo profile, con l'effetto di generare una classe a runtime creata ereditando fa **ProfileCommon** ed aggiungendo le proprietà definite dallo sviluppatore.

```
<configuration>
  <sytem.web>
    <profile enabled="true">
      <properties>
        <add name="FullName" type="string" />
      </properties>
    </profile>
  </sytem.web>
</configuration>
```

Attraverso il nodo group è anche possibile aggiungere dei gruppi di proprietà, così che sia più semplice raggrupparle. VS 2005 è in grado in automatico di elencare le proprietà anche se in realtà queste verranno rese persistenti in una classe solo alla successiva esecuzione. Questa caratteristica rende semplice l'utilizzo delle Profile API. A meno che non venga abilitato (e di default non lo è), il salvataggio delle proprietà nel profilo va fatta manualmente, invocando il metodo Save.
Un tipico esempio che crea il profilo ed imposta alcune proprietà potrebbe essere il seguente:

```
ProfileCommon.Create(User.Identity.Name);
Profile.FullName = "Daniele Bochicchio";
Profile.Save();
```

Nel caso dell'accesso alle informazioni del profilo, è sufficiente utilizzare direttamente la proprietà Profile, esposta da Page o HttpContext. L'accesso alle informazioni sarà rapido e strongly-typed, come già detto. Ovviamente all'interno del profilo possono essere salvate tutte le informazioni di cui si ha bisogno, anche se questo è vero a patto che il tipo di provider utilizzato sia flessibile da poter garantire tutto questo.

CONCLUSIONI

La combinazione di queste tre API, basate sul provider model, rende possibile la creazione di applicazioni protette, con supporto per ruoli e profili senza particolari sforzi. Quello che può essere ancora sottolineato è che il tutto si applica tranquillamente a qualsiasi tipo di provider, non solo a quello per SQL Server che è stato analizzato nello specifico in questo articolo, e che nulla ci vieta di utilizzare provider diversi per le diverse API. Anche se, probabilmente, l'unica volta in cui li userete in maniera differente sarà quando avrete bisogno di personalizzarne uno.

Daniele Bochicchio

Daniele Bochicchio è il content manager di ASPItalia.com, community che si occupa di ASP.NET, Classic ASP e Windows Server System. Il suo lavoro è principalmente di consulenza e formazione, specie su ASP.NET, e scrive per diverse riviste e siti. È Microsoft ASP.NET MVP, un riconoscimento per il suo impegno a supporto delle community e per l'esperienza maturata negli anni. Il suo blog è all'indirizzo http://blogs.aspitalia.com/daniele/

JAVA FOR ENTERPRISE ARRIVA SEAM!

SEAM È IL FRAMEWORK CHE SI APPRESTA A RIVOLUZIONARE LO SVILUPPO DI APPLICAZIONI J2EE SEMPLIFICANDO LA SCRITTURA DEL SOFTWARE TRAMITE L'USO DI TECNICHE AVANZATE QUALI JSF E COMPONENTI SERVER COME EJB E POJO

Java Enterprise Edition è sempre stato un ambiente piuttosto ostico da padroneggiare. Ultimamente si vede un forte interesse per la semplificazione delle specifiche (per esempio la creazione di EJB3 è resa notevolmente più semplice rispetto al passato) e c'è la costante evoluzione degli IDE, sempre più evoluti (e con wizard per la creazione automatica di parte dell'applicazione). Eppure quello che mancava era un framework che davvero semplificasse la scrittura delle applicazioni pur utilizzando gli standard esistenti. Seam è la risposta a questa esigenza. Esso è, innanzi tutto, un framework per ambienti Java EE 5 e possiede numerose caratteristiche che facilitano l'uso di tecnologie standard, comprese alcune tra le tecnologie emergenti di maggior interesse. Infatti il framework permette di utilizzare, in maniera uniforme, diversi tipi di componenti di business (sia classi POJO gestite con Hibernate o altri tool ORM, componenti EJB 3.0 o semplici classi che usano le primitive JDBC), integrare agevolmente la tecnologia JSF per la parte di presentazione, creare applicazioni che possono far uso di AJAX. Particolarmente semplice è anche la creazione di workflow attraverso l'uso di un business process management denominato jBPM; molte caratteristiche sono supportate da regole che si basano sul modello "dichiarativo", grazie all'uso delle annotazioni (introdotte nel linguaggio Java dalla versione 5.0). Anche la gestione dei contesti di un'applicazione Web è stata semplificata ed estesa (in particolare, oltre ai "classici" contesti di una servlet - request, session e application – Seam permette di specificare due ulteriori contesti: conversation e business process).

Molte altre caratteristiche risultano utili per chi sviluppa: possibilità di creare test in maniera semplice sia per i test di modulo che per quelli di integrazione (in particolare è consigliato l'uso del tool TestNG, ma è supportato, ovviamente, anche JUnit). Nel seguito i dettagli per l'installazione e l'uso del framework.

PRIMA DI INIZIARE

Prima di installare e usare Seam è necessario avere installato il JDK versione 5.0 (o 1.5.0 che dir si voglia) e almeno un Servlet Container (come Tomcat) o, preferibilmente, un Application Server; ant è necessario per la compilazione e il deploy dei sorgenti degli esempi. Nell'articolo si farà uso di JBoss AS 4.0.4 GA. È importante che quest'ultimo sia installato usando il profilo EJB3 (tale scelta avviene in fase di installazione, come si può vedere dalla Figura 1; per informazioni sull'ultima release di JBoss AS si può consultare la pagina http://www.jboss.org/products/jbossas).

Accertato che l'ambiente iniziale contenga tutti gli strumenti, si può procedere al download dell'ultima versione di Seam (la 1.0.1GA al momento di scrivere l'articolo); un'utile "roadmap" su come intallare il framework è reperibile alla pagina http://labs.jboss.com/portal/jbossseam/gettingstarted. Seam si presenta come un archivio compresso che va esploso sulla propria macchina. Analizziamone la struttura.

LA STRUTTURA DI SEAM

Le principali cartelle sono doc/, dovè possibile reperire la documentazione del framework (sono presentate sia le API che mette a disposizione sia una guida di riferimento che spiega come partire per scrivere le applicazioni), src/ che contiene tutti i sorgenti (infatti il framework è reso disponibile con licenza LGPL) ed examples/, che contiene alcuni esempi d'uso.

GLI ESEMPI DI SEAM

Seam viene fornito con numerosi esempi: vere e proprie applicazioni pronte ad essere deploytate, usate e analizzate per comprendere alcune regole di scrittura ottimali usando il framework.
È interessante notare che ciascun esempio fa uso di EJB e, pertanto, dovrebbe essere installato su un application server. In realtà sono installabili anche su un servlet container e, in quest'ultimo caso, l'applicazione viene installata con un particolare container per EJB3: JBoss Embeddable EJB3 (in pratica un container completamente "incapsulato" dentro l'applicazione).
Prima di eseguire ant per la compilazione e il deploy degli esempi è necessario editare il file build.properties (presente nella cartella di installazione di seam). In esso vanno inseriti i path corretti dove sono installati gli ambienti (in tomcat.home si indichi la home di Tomcat; se si usa JBoss si specifichi la sua home nel parametro jboss.home). Per eseguire il deploy di un esempio posizionarsi su di esso con una console di comandi (per esempio, per installare booking, posizionarsi sotto examples/booking) e, dopo aver fatto partire l'Application Server, digitare il comando:

```
ant deploy
```

A questo punto se si accede alla URL http://localhost:8080/seam-booking/ si vedrà l'applicazione in esecuzione.

UNA NUOVA APPLICAZIONE

Dopo aver "curiosato" tra gli esempi proposti si può procedere alla creazione di una nuova applicazione. In particolare verrà mostrato come creare un front-end per la consultazione delle riviste di ioProgrammo. La base dati sarà composta da un'unica tabella: riviste. Essa contiene alcuni numeri della rivista con i seguenti campi: **numero**, **strillo** (ovvero il titolo principale) e **urlimmagine** (dove reperire l'immagine della copertina); numero è la chiave univoca. Si mostreranno i diversi numeri usando una paginazione dei risultati, e si farà in modo che chi visita il sito possa, posizionandosi con il mouse su un elemento della lista, visualizzare la copertina del numero. I dati vengono reperiti dal DB usando degli EJB; la presentazione avviene con le pagine JSF.
Come cominciare? Per esempio si potrebbe eseguire una copia di uno degli esempi contenuti in Seam e personalizzarla (usandola come template). Questo aiuta a mantenere una struttura del tutto simile agli altri esempi e a riusare le parti di configurazione già impostate. Si faccia una copia dell'applicazione examples/messages in examples/ioprogrammo. Le cartelle view/ ed src/ vanno svuotate (esse conterranno, rispettivamente, le JSF di visualizzazione e gli EJB della parte di backend). I file build.xml e readme.txt vanno personalizzati (si veda il codice allegato per i dettagli); infine la cartella resources/ verrà riusata, per ora, così com'è. Da questo "scheletro" realizziamo l'applicazione…

UN EJB PER LE RIVISTE

Si può creare un EJB come si creerebbe una qualsiasi classe JavaBean. Nel caso di una rivista, si dovrebbero dichiarare gli attributi e i metodi **setter/getter** opportuni:

```
public class Riviste implements Serializable{
  private Long numero;
  private String strillo;
  private String urlimmagine;
  public Long getNumero() {
    return numero;
  }
  public void setNumero(Long numero) {
    this.numero = numero;
  }
 // Altri setter/getter
}
```

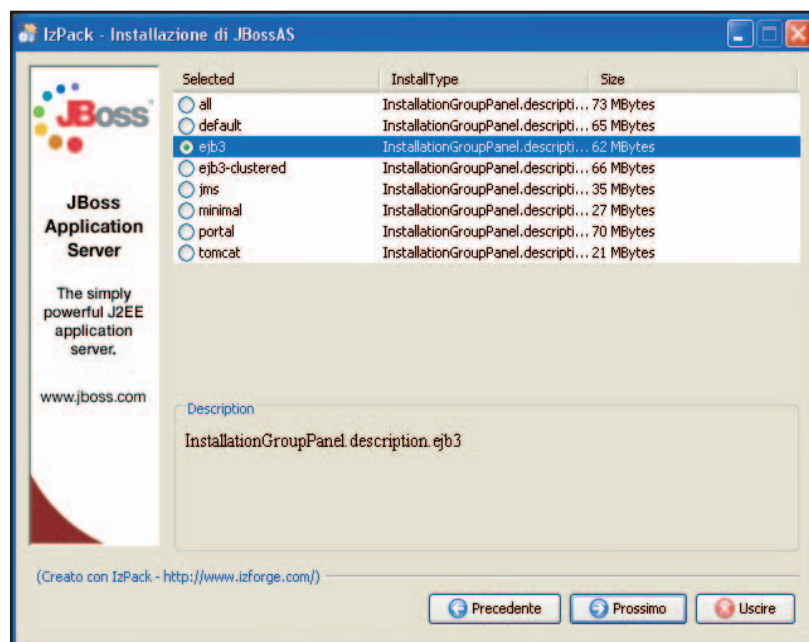



Fig. 1: scelta del profilo EJB3 in fase di installazione di JBoss AS 4.0.4.

Ovviamente questo è proprio un JavaBean! Per farlo diventare un EJB è necessario dichiararlo usando un'apposita annotazione. È possibile creare tre tipi di EJB: entity, stateless, stateful. Lo si fa inserendo una delle seguenti annotazioni prima della dichiarazione della classe:

```
@Entity
@Stateless
@Stateful
```

Un EJB di tipo entity non dovrebbe né accedere al database né effettuare alcun tipo di transaction managment; uno di tipo stateless non ha queste limitazioni ma non mantiene il suo stato tra due interazioni successive. Uno di tipo stateful mantiene lo stato tra richieste successive al server.
Per il bean Riviste si dovrà inserire @Entity prima della dichiarazione della classe.

COMPONENTI DI SEAM

Ogni componente di Seam deve essere marcato come tale usando la annotazione @Name("suoNome") (e il nome associato deve essere univoco all'interno dell'applicazione). Per il bean Riviste si inserisca l'annotazione @Name("riviste") prima della dichiarazione della classe.
Esistono poi una serie di annotazioni utili per il controllo dei valori; per esempio ecco come dichiarare che il "numero" è un identificativo sulla base dati:

Fig. 2: l'esecuzione di un'applicazione di esempio (seam-booking).

```
@Id
public Long getNumero() {
  return numero;
}
```

Quando la chiave è un progressivo (ovvero il suo valore viene autogenerato dal database) lo si indica con l'annotazione @Generated Value. Ecco invece come indicare che la lunghezza dello strillo deve essere inferiore a 100 caratteri e non nulla:

```
@NotNull @Length(max=100)
public String getStrillo() {
  return strillo;
}
```

In modo simile è possibile personalizzare gli altri metodi getter. Non resta che definire un componente che si occupa del reperimento delle riviste dalla base dati e le passi alla pagina di presentazione. Esso è un bean di tipo stateful, con scope di sessione e a cui si può assegnare il nome "rivisteManager":

```
@Stateful
@Scope(SESSION)
@Name("rivisteManager")
public class RivisteManagerBean
 implements Serializable, RivisteManager {
 // implementazione...
}
```

Si noti che tale EJB implementa un'interfaccia: essa non fa altro che definire i metodi che il bean implementerà. La realizzazione dipende da come si vogliono presentare i dati. Per capirlo analizziamo la pagina JSF che mostrerà i dati...

LA PAGINA JSF DI PRESENTAZIONE

Si crei una nuova pagina, riviste.jsp. Al suo interno ci saranno sia tag JSF che tag specifici di Seam. Per questo motivo è necessario cha la pagina venga "interpretata" da un apposito gestore di Seam: esso legge le url con estensione .seam e ne cerca il corrispettivo .jsp (un po' come accade per le JSF standard). Per far sì che chi si collega alla webapp apra automaticamente la url riviste.seam (che, ricordiamo, consiste nell'invocare Seam sulla pagina riviste.jsp), si può creare un semplice file index.html con i seguenti tag:

```
<html>
```

```
<head>
 <meta http-equiv="Refresh" content="0;
                           URL=riviste.seam">
</head>
</html>
```

In pratica c'è un redirect sulla pagina voluta. Quest'ultima conterrà, all'inizio, le usuali dichiarazioni d'udo delle librerie; per le JSF:

```
<%@ taglib uri="http://java.sun.com/jsf/html"
                              prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core"
                              prefix="f" %>
```

Inoltre essa può contenere la dichiarazione dei componenti esposti da Seam:

```
<%@ taglib
    uri="http://jboss.com/products/seam/taglib"
                              prefix="s" %>
```

Come al solito la parte "dinamica" della pagina JSF è racchiusa dal tag <f:view>…</f:view>. In esso potremmo, per esempio, suddividere la pagina in due usando un panelGrup con due colonne:

```
<h:panelGrid columns="2" border="1">
...
</h:panelGrid>
```

Nella prima colonna potremmo inserire la lista delle riviste, nella seconda l'immagine della copertina. La lista può essere realizzata usando uno dei componenti più flessibili delle JSF: la dataTable; essa dovrà dichiarare in "value" il valore da cui legge i dati (tale valore deve essere una lista di elementi), in "var" la variabile con cui si referienzia ogni singolo elemento, più eventuali altri attributi:

```
<h:dataTable
 var="riv"
 value="#{rivisteList}"
 width="400">
...
</h:dataTable>
```

C'è da capire come i componenti vengano referenziati dalle pagine JSF; infatti se in una pagina JSF si fa uso di una variabile (per esempio rivisteList) allora Seam verifica se esiste un componente con lo stesso nome; se sì, e se la variabile non è stata inizializzata, Seam si preoccupa di istanziare il componente, e associa la nuova istanza creata alla variabile nella pagina JSF.

Ma chi è rivisteList? Esso dovrà essere un oggetto di tipo List al cui interno sono memorizzati i singolo EJB di tipo Rivista. Chi lo dovrà esporre sarà uno dei componenti creati: RivisteManagerBean. Vedremo nel seguito come. Per ora concentriamoci su cosa si scrive all'interno del dataTable; volendo inserire su una colonna i numeri della rivista e su un'altra lo strillo, si potrebbe scrivere:

```
<h:column>
  <f:facet name="header">
    <h:outputText value="Numero"/>
  </f:facet>
  <h:outputText value="#{riv.numero}"/>
</h:column>
<h:column>
  <f:facet name="header">
    <h:outputText value="Strillo principale"/>
  </f:facet>
  <h:outputText value="#{riv.strillo}" />
</h:column>
```

Si noti come da "riv" si faccia riferimento agli attributi dell'entity bean (in realtà ciascun attributo è accessibile perché esposto dal rispettivo metodo getter). La seconda colonna avrà un semplice riferimento ad un'immagine:

```
 <h:graphicImage url="img/nessuna.jpg"
                     width="300" height="410" />
```

Resta da fare in modo che quando l'utente si posiziona sopra uno strillo, l'immagine sia quella della relativa copertina. Un modo per farlo è ricorrere ad un effetto di rollover javascript; pertanto si modificherà la stampa dello strillo scrivendo:

```
<h:outputText value="#{riv.strillo}"
onmouseover="document.images[0].src='#
                         {riv.urlimmagine}'"
onmouseout="document.images
                 [0].src='img/nessuna.jpg'"  />
```

RIEMPIRE LA DATATABLE

Resta il problema di modificare Riviste ManagerBean affinché riempia un'apposita lista da passare al componente dataTable; per farlo si dichiari, con l'annotazione @Data Model, qual è l'oggetto che memorizza i dati:

```
@DataModel
 private List<Riviste> rivisteList;
```

Poi si realizza un metodo che esegue la query di reperimento dei valori sul DB (si noti l'uso della annotazione @Factory per indicare che il metodo è quello che inizializza rivisteList):

```
@Factory("rivisteList")
 public void findRiviste() {
   rivisteList = em.createQuery(
    "from Riviste rv order by rv.numero"
   ).getResultList();
}
```

Ecco spiegato come la JSF possa riferisi a rivisteList nei propri tag! em è un oggetto di tipo EntityManager dichiarato in questo modo:

```
@PersistenceContext(type=EXTENDED)
 private EntityManager em;
```

Questo è quanto! Ora l'applicazione è pronta per essere deploytata e testata. Ma prima ci si può chiedere… e il database?

DA CHE PARTE PER IL DATABASE

Come avviene la creazione e, soprattutto, dov'è indicato il database da usare? Esso è nel file resources/META-INF/persistence.xml:

```
<persistence>
 <persistence-unit name="messageDatabase ">
  <provider>
   org.hibernate.ejb.HibernatePersistence
   </provider>
   <jta-data-source>
    java:/DefaultDS
   </jta-data-source>
   <properties>
    <property name="hibernate.hbm2ddl.auto"
      value="create-drop"/>
   </properties>
  </persistence-unit>
</persistence>
```

Fig. 3: l'applicazione in esecuzione.

Benché si sia mantenuto come nome messageDatabase, è bene crearne uno nuovo (per esempio IoProgrammoDatabase); altrimenti ci possono essere conflitti con il db usato dall'esempio "message". L'attributo jta-data-source indica che il db in uso è preso dai data-source dell'Application Server (è quello di default). Invece specificando che la proprietà hibernate.hbm2ddl.auto ha valore create-drop si fa sì che il db venga creato all'avvio dell'applicazione e distrutto quando essa termina. Questa proprietà è utile solo in fase di test. I dati con cui va inizializzata la base dati vanno inseriti nel file resources/import.sql (nell'esempio ci sono varie insert sulla tabella Riviste).
La tabella Riviste non viene mai definita; infatti essa viene creata a partire dall'omonimo EJB (i nomi dei campi e i loro tipi sono gli stessi degli attributi dell'EJB!).

INSERIRE LA PAGINAZIONE

Ora che il primo scheletro dell'applicazione è funzionante si può inserire la paginazione della lista delle riviste. Per farlo è necessario sia indicare dei nuovi componenti nella pagina JSF, come:

```
<s:link value="Pagina precedente"
             action="#{rivisteManager.prevPage}" />
<s:link value="Pagina successiva"
              action="#{rivisteManager.nextPage}"/>
```

Sia realizzare le azioni ivi indicate (ovvero nextPage e prevPage); tali azioni saranno realizzate sempre all'interno di RivisteManager Bean; è opportuno definire nuovi attributi per la gestione della paginazione (actualPage, che è la pagina attuale, pageSize che è la dimensione di ogni pagina e, infine, maxPages che indica il numero massimo di pagine); inoltre dovrà essere modificato il metodo findRiviste affinché reperisca solo gli elementi necessari:

```
public void findRiviste() {
  rivisteList = em.createQuery(
   "from Riviste rv order by rv.numero"
  ).getResultList();
  maxPages =
    rivisteList.size() / pageSize  ;
  rivisteList = rivisteList.subList(
    actualPage*pageSize,
    Math.min(
      rivisteList.size(),
     (actualPage+1)*pageSize
    )
  );
}
```

Si potrà notare come la query reperisca tutti gli elementi e poi, dalla lista, vengano mantenuti solo quelli necessari. Per ottimizzare si potrebbe limitare la query affinché reperisca i dati a partire da un certo record (metodo setFirstResult(numero)) e per un massimo numero di record (setMaxResults(quanti)). Ecco, infine, le azioni di cambio pagina (ciascuna delle quail invoca findRiviste per ripopolare la lista da mostrare):

```
public void nextPage() {
 if (actualPage<maxPages)
   actualPage++;
 this.findRiviste();
}
public void prevPage() {
 if(actualPage>0)
   actualPage--;
 this.findRiviste();
}
```

Il deploy, come per gli altri esempi, viene fatto con il comando:

```
ant deploy
```

In Figura 3 l'applicazione completa e funzionante, accessibile dall'url http://localhost:8080/seam-ioprogrammo/.

I FILE DI CONFIGURAZIONE

Può sembrare sorprendente, ma non è necessario personalizzare alcun altro tipo di codice o file. Infatti, se si va a curiosare nella cartella resources/, in essa sono indicati dei file di configurazione del tutto generali e riutilizzabili per qualsiasi applicazione Seam. In particolare non esiste alcuna dichiarazione di asscociazione tra i componenti EJB e le JSF! Tutti i riferimenti vengono risolti al runtime.

E LA GESTIONE DEI DATI?

Resterebbero da creare le pagine di data-entry per la gestione dei dati nel DB. Benché si possa procedere come mostrato in precedenza, in realtà è possibile ricorrere anche alla generazione automatica usando gli Hibernate Tools di Eclipse! Tali tool sono scaricabili dalla pagina http://tools.hibernate.org/ e si preoccupano di fare il reverse engineering di una base dati esistente. A parte la generazione automatica dei file di configurazione per Hibernate, è possibile specificare la creazione di uno scheletro di un'applicazione completa usando Seam. Tale applicazione è comprensiva di tutte le parti di gestione del data-entry dei dati!

CONCLUSIONI

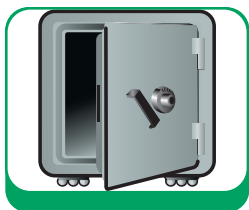
Prima di usare Seam è consigliabile approfondire la conoscenza architetturale del modello J2EE. Infatti, benché Seam semplifichi l'interazione delle diverse tecnologie (JSF per la presentazione ed EJB per la logica di business), resta fondamentale avere di esse una conoscenza, anche minima.
Essendo un framework ancora relativamente "giovane" presenta comunque notevoli punti di forza e ottime potenzialità. Non resta che attendere un adeguato supporto da parte dei più diffusi IDE e di sicuro esso diverrà parte fondamentale del bagaglio di conoscenze di uno sviluppatore di applicazioni J2EE.

Ivan Venuti

JBOSS E REDHAT

JBOSS Application Server si conferma come il Server Leader per le applicazioni J2EE sul mercato. Supporta interamente le specifiche 1.4 così come funzionalità di caching , clustering, e persistenza. Include il supporto per EJB 3.0 che consente un rapido sviluppo di applicazioni di tipo Enterprise.
Da poco JBOSS è diventata una divisione di Red Hat e si occupa dello sviluppo del MiddleWare per applicazioni Java. Oltre a JBOSS produce e fornisce assistenza per alcuni prodotti piuttosto noti quali ad esempio Hibernate. Nel campo degli Application Server rimane comunque il leader indiscusso del mercato. Non male per un progetto nato in sordina e iniziato con una forte propensione all'OpenSource. Il sito di riferimento per tutti i progetti legati a JBOSS è http://www.jboss.org

IDENTITÀ DIGITALE COME GESTIRLA?

CHI SI NASCONDE DIETRO AL BROWSER? NELLA VITA REALE È SEMPLICE ESIBIRE UN DOCUMENTO CHE CERTIFICHI L'IDENTITÀ. MA COME FARE QUANDO IN RETE È NECESSARIO ESSERE CERTI IN MODO DEFINITIVO DEI DATI DI UN UTENTE?

La questione dell'identità digitale su internet è un problema ancora aperto. Identificarsi correttamente ed affermare la propria identità in modalità sicura è un problema attualmente risolvibile solo con l'ausilio di cerficati digitali o di smart card. Nel mondo reale abbiamo a disposizione diversi modi per affermare la nostra identità. Ad esempio la carta di credito mi identifica inequivocabilmente nel momento in cui effettuo un qualsiasi pagamento fornendo il mio numero identificativo. La mia patente di guida mi identifica come un qualsiasi cittadino abilitato alla guida mostrando informazioni come il numero di patente e la data di scadenza. La mia carta di identità, invece, mi identifica come un cittadino italiano fornendo informazioni personali come il mio stato civile o la mia professione, oltre al nome e all'indirizzo. In base a queste considerazioni ci accorgiamo che contesti diversi richiedono informazioni diverse. Se dobbiamo effettuare un pagamento, poco importa il mio numero di patente, o viceversa. Allo stesso modo, nel mondo digitale, il concetto non può che essere lo stesso: differenti tipi di identità. Questo concetto, però, aggiunge una complessità attualmente non gestita, ma probabilmente necessaria.

L'IDENTITÀ DIGITALE

Gestire un problema come l'identità digitale non può essere un affare di una singola organizzazione. La creazione di diverse identità può riguardare diverse organizzazioni. Pensiamo ad esempio alla banca che fornisce un ipotetica card per accedere al nostro conto online, oppure al comune della città che crea una card digitale per accedere ai servizi online. Così come nel mondo reale, anche nel mondo digitale abbiamo bisogno di differenti card che forniscono differenti informazioni in differenti contesti. Questo meccanismo non può essere prerogativa di un'unica organizzazione poiché prevede, appunto, il coordinamento di più di un fornitore di identità (identity provider). Ecco che nasce l'esigenza di mettere a punto un sistema coordinato per la gestione delle identità, comunemente chiamato identity metasystem. In questo sistema l'identità è rappresentata da un security token, un set di claims ognuno dei quali fornisce una parte delle informazioni che compongono l'identità stessa. Un claim rappresenta, quindi, una singola informazione come la username, l'indirizzo di casa, l'indirizzo e-mail, ma anche dati sensibili come, ad esempio, il numero della carta di credito. L'utilizzo di un identity metasystem consente la creazione di più identità ognuna delle quali fornisce un security token che, attraverso la gestione dei claims richiesti, garantisce l'autenticazione dell'utente. Ovviamente possiamo riutilizzare ogni identità tutte le volte che ne abbiamo bisogno fornendo la relativa card a quei siti o a quelle applicazioni che ne fanno richiesta.

Il coinvolgimento di più organizzazioni richiede, però, l'implementazione di una soluzione unica ed indipendente da un singolo vendor, che non sia basata, quindi, su protocolli di comunicazione proprietari. Partendo da questa considerazione, Microsoft ha creato con Windows CardSpace (WCS) un sistema facilmente estendibile proprio perché basato su protocolli standard come SOAP e XML, oltre all'uso di WS-Security, WS-SecurityPolicy, WS-Trust e WS-Metadata Exchange.

WINDOWS CARDSPACE: UNA SOLUZIONE UNICA

Con WCS qualsiasi applicazione Windows può interagire con un identity metasystem e fornire le card come richiesto dall'utente. Ma come è possibile farlo anche con le applicazioni web? Un browser, come Internet Explorer, è comunque un'applicazione Windows che sfrutta, quindi, tutte le potenzialità offerte da questo suo status. Oltre al browser di casa Microsoft sono comun-

que già pronte implementazioni per Mozilla FireFox e Safari.
L'identity metasystem prevede la partecipazione di tre principali entità:

- User: è l'entità associata ad una specifica identità digitale. Sono per la maggiore persone, ma anche le organizzazioni, le applicazioni, le macchine o altre entità possono avere identità digitali;
- Identity provider: un identity provider è l'entità che fornisce una identità digitale per un utente;
- Relying party: è un'applicazione che in qualche modo accetta un'identità digitale. Una relying party autenticare ed eventualmente autorizza un utente proprio attraverso l'accettazione di una identità digitale proprio come avrebbe fatto nel caso di un form di login;

Un identity provider fornisce l'identità allo user per l'autenticazione verso un relying party. Ad esempio, per l'identità digitale fornita dal proprio datore di lavoro l'identity provider è un sistema come Active Directory. In un sito internet qualsiasi, invece, l'identity provider è lo stesso utente poiché è lui che definisce ed accerta i propri dati. Una relying party, invece, può essere un qualsiasi sito internet che necessita di identificare i propri utenti, o qualsiasi applicazione che accetta richieste via web services. Nell'articolo vedremo come creare un form che accetta una identity card fornita con WCS e vedremo come creare un servizio che si autentica utilizzando la stessa card.

CREIAMO LE NOSTRE IDENTITY CARD CON WCS

Con l'installazione del .NET Framework 3.0 nella versione June CTP, utilizzata nel presente articolo, nel pannello di controllo viene aggiunto un nuovo item che consente l'accesso all'Identity Manager o Identity Selector attraverso l'icona Windows CardSpace, come in figura 1.
Per creare una nuova card è sufficiente cliccare sul pannello di destra alla voce Add a Card oppure sulla card vuota visualizzata all'interno dell'elenco delle card. Successivamente ci viene chiesto quale tipo di card dobbiamo creare. Abbiamo a disposizione due tipologie:

- Personal card: come riportato nel wizard, una personal card viene utilizzata per l'identificazione in siti e servizi online, affiancandosi o in alcuni casi sostituendo le classiche username e password;
- Managed card: sono le card emesse da organizzazioni, come ad esempio le banche;

la sostanziale differenza tra le due tipologie riguarda la responsabilità dei dati. Nel primo caso siamo noi i responsabili dei dati dichiarati, mentre nel secondo caso la responsabilità è a carico del fornitore della card (identity provider). In questo caso, quindi, subentra una terza entità che certifica l'identità dell'utente. A garanzia e a supporto dell'identità, alla card può essere associato anche un certificato o una smart card. In breve abbiamo quattro tipologie di managed card:

- Una card che utilizza l'autenticazione tramite username e password;
- Una card che utilizza l'autenticazione su protocollo Kerberos;
- Una card che utilizza un tipo di autenticazione proprietario basato su un identificativo;
- Una card che utilizza una autenticazione basata su una smart card o un certificato;

la creazione di una personal card, invece, prevede il semplice inserimento di un set predefinito di informazioni che andranno a formare i claims del relativo security token. I dati prevedono l'inserimento del nome, del cognome, dell'indirizzo, ecc … oltre all'assegnazione di un nome alla stessa card.
WCS si compone di una applicazione client, chiamato identity selector, che si preoccupa di recuperare ed inviare le card verso il server.

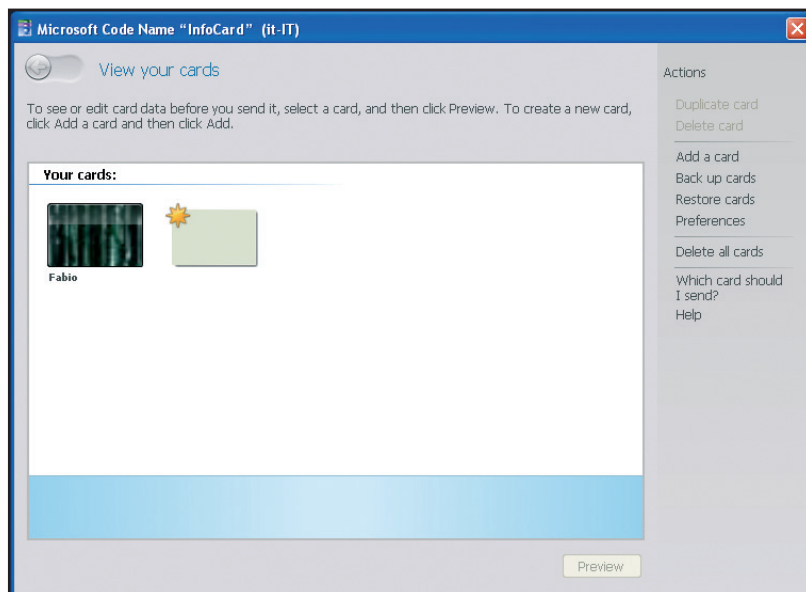



Fig. 1 *L'Identity Manager o Identity Selector per la creazione delle card.*

AUTENTICARSI SU UN SITO WEB

La card creata può essere utilizzata in qualsiasi ambito per il riconoscimento e l'autenticazione. Immaginiamo di avere la possibilità di registrarci ed autenticarci sul sito ioProgrammo utilizzando una card creata con CardSpace. Il processo di registrazione della card nell'applicazione web avviene in due fasi:

1. **Associazione** di un account utente con una specifica card
2. **Autenticazione** dell'utente utilizzando la card

La prima fase può essere completata contestualmente alla registrazione dell'utente. Presentando la card, infatti, si può eventualmente solo scegliere il nome dell'account da associare all'utente. Per motivi di spazio in questo articolo non tratteremo questa parte del processo, ma verificheremo come autenticare un utente utilizzando una carta creata con WCS. Nell'atto di associazione della card, associamo all'account utente l'identificativo univoco che avremo ottenuto. La pagina di autenticazione offrirà il classico form basato su username e password, e in alternativa il form di autenticazione basato sulla selezione di una card, simile a quello visualizzato in figura 3.
Per l'esempio utilizziamo Internet Explorer 7 beta 3 (IE7) perché attualmente è l'unico browser che supporta l'inserimento della card tramite l'utilizzo del Identity Selector. Sono però già in fase di sviluppo plugin per FireFox e Safari.
Per ottenere il token dalla card inserita utilizzeremo un tag object che istanzia nella pagina un

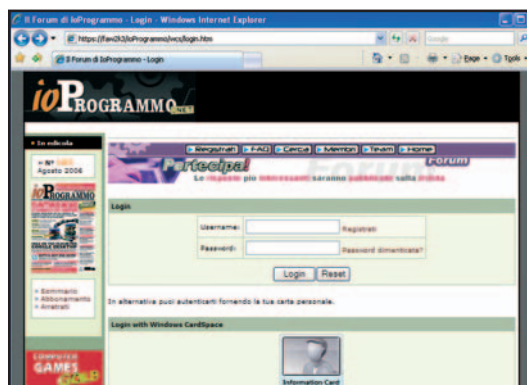



Fig. 2: La pagina di autenticazione di ioProgrammo

tipo application/x-informationcard, ottenendo un codice html simile a questo:

```
<object type="application/x-informationcard"
                                   name="xmlToken">
   <param name="tokenType"
  value="urn:oasis:names:tc:SAML:1.0:assertion" />
   <param name="issuer"
 value="http://schemas.microsoft.com/ws/2005/05/id
                             entity/issuer/self" />
   <param name="requiredClaims"
 value="http://schemas.microsoft.com/ws/2005/05/id
                         entity/claims/givenname,
http://schemas.microsoft.com/ws/2005/05/identity/cl
                                    aims/surname,
http://schemas.microsoft.com/ws/2005/05/identity/cl
                               aims/emailaddress,
http://schemas.microsoft.com/ws/2005/05/identity/cl
                 aims/privatepersonalidentifier" />
</object>
```

Per attivare l'Identity Selector e successivamente l'invio del token, utilizziamo quattro parametri particolari, così come evidenziato nella seguente tabella:

type="application/x-informationcard"	Dichiariamo il tipo di object nel flusso html per abilitare il browser allo scambio dei dati con esso
param name= "tokenType"	Con questo parametro impostiamo il tipo di token che devo utilizzare, che in questo caso sarà SAML 1.0
param name= "issuer"	L'URL dell'Identity Provider. Nel nostro esempio, però, si tratta di un URI che invoca l'Identity Provider per l'auto generazione della card
param name= "requiredClaims"	Quest'ultimo parametro elenca I claims richiesti dalla Relying Party per questa particolare autenticazione

IE7 esegue l'Identity Selector direttamente allo scatenarsi dell'evento submit del form in cui esso è incluso, utilizzando come parametri il contenuto del tag object con cui esso interagisce. È opportuno evidenziare che al fine di garantire una corretta e sicura comunicazione tra il client ed il server è necessario utilizzare una connessione sicura costruita con il protocollo SSL. Alla versione rilasciata al momento della stesura del presente articolo (July CTP) questo costituisce un requisito fondamentale. L'Identity Selector permette la selezione della card, la stessa utilizzata al momento della registrazione, e l'invio al server, come visualizzato in figura 4.
Dopo l'invio della card dal lato server otteniamo il token di autenticazione. Qui la trattazione dei

dati diventa a carico nostro. Utilizzando le API fornite da System.ServiceModel e dal System. IdentityModel riusciamo ad estrarre dal token, in formato SAML, i claims richiesti. Per questa operazione ci viene in aiuto la classe TokenHelper,

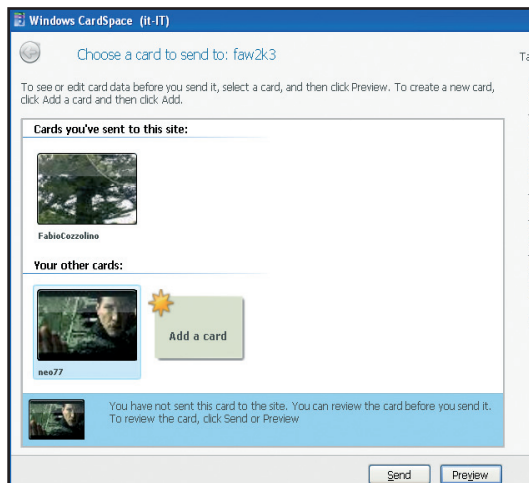



Fig. 3: ***La selezione della card per ioProgrammo.***

un helper che astrae l'uso delle funzioni del ServiceModel di WCF e disponibile sul sito wcs.netfx3.com.
Utilizzando il metodo GetUniqueID del Token Helper recuperiamo il PPID (Private Personal Unique Identifier) un identificatore che identifica l'utente univocamente nei confronti di una relying party. Questo significa che viene generato un PPID per ogni specifica Relying Party e che tutti i PPID generati non sono tra loro collegati. Allo stesso modo l'Identity Selector utilizza il PPID per verificare se una specifica card è stata già inviata o meno al Relying Party. Il PPID è generato dall'Identity Provider che, nel caso di una self-issued card, una carta emessa da noi, è generata dal self-issued provider.

Fig. 4: ***L'autenticazione avvenuta sul sito di ioProgrammo***

AUTENTICARSI SU UN WEB SERVICE

Allo stesso modo, così come abbiamo utilizzato la card creta per autenticarci sul sito ioProgrammo, usiamo la stessa card con un web service. Ipotizziamo di voler utilizzare i servizi esposti da un bookstore per ricercare un libro. Per ottenere l'autenticazione in modo sicuro utilizzeremo la card, sempre unitamente alla connessione sicura garantita dall'uso del certificato su protocollo HTTPS, che accompagnerà il messaggio di richiesta. Costruiamo il servizio utilizzando Windows Communication Foundation (WCF) come framework per la gestione delle richieste integrato con il web server IIS. Il codice completo del servizio è disponibile nel cd allegato alla rivista. Per motivi di spazio tratteremo solo le modifiche che riguardano il file di configurazione. In WCS, Windows CardSpace corrisponde alla tipologia di credenziali "IssuedToken". Apriamo quindi il file .config e aggiungiamo una sezione di binding come la seguente:

```
<bindings>
  <wsHttpBinding>
    <binding name="bookSearchBinding">
      <security mode="Message">
        <message
                clientCredentialType="IssuedToken" />
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

La stessa modifica deve poi essere replicata nel file di configurazione dell'applicazione client. Così come abbiamo visto per l'autenticazione su una web application, anche l'autenticazione su WCF richiede l'utilizzo di un certificato, di tipo X.509, per

PERCHÉ CARDSPACE NON È PASSPORT

Analizzando le caratteristiche proprie del modello proposto da WCS (Windows CardSpace) non possiamo ignorare la similitudine con il sistema Microsoft Passport. In realtà le differenze sono molte, tantissime. Passport consente l'autenticazione su più applicazioni web e web services utilizzando un unico account utente. Una buona idea che però non ha avuto il seguito necessario per gli alti costi di sottoscrizione e per il sistema chiuso su quale è tuttora basato.
Il sistema WCS, invece, è basato su protocolli standard come SOAP, XML e SAML. La sua infrastruttura mira a superare l'autenticazione debole costruita su username e password, fornendo invece delle information card che rappresentano l'utente. Il sistema è completamente aperto ad implementazioni esterne proprio grazie all'uso dei protocolli sopracitati. Passport, inoltre, diventerà un identity provider per CardSpace, ma CardSpace non dipende affatto da Passport, il quale non avrà nessun ruolo particolare nell'identity metasystem.

il riconoscimento del server, emesso da autorità di certificazione come VeriSign. WCF and CardSpace utilizzeranno i certificati per criptare e firmare i tokens di sicurezza. In questa ottica modifichiamo i files di configurazione per recuperare le informazioni del certificato aggiungendo il seguente behavior:

```
<behaviors>
<serviceBehaviors>
        <behavior
            name="bookSearchBehavior"
  returnUnknownExceptionsAsFaults="true">
<serviceMetadata httpGetEnabled="true" />
<serviceCredentials>
<serviceCertificate
                findValue="faw2k3"
storeLocation="LocalMachine"
                storeName="My"
x509FindType="FindBySubjectName"/>
        </serviceCredentials>
        </behavior>
</serviceBehaviors>
</behaviors>
```

In questo modo identifichiamo il certificato FAW2K3 tra le fonti attendibili sulla macchina locale. Ora colleghiamo la configurazione del servizio al behavior creato aggiungento l'attributo behavorConfiguration="bookSearchBeha- vior". Infine abbiamo la necessità di garantire l'accesso a CardSpace al certificato specificando l'apposito riferimento all'interno dell'endpoint. Modifichiamo quindi la sezione services in questo modo:

```
<services>
<service name="Library.BookSearch"
behaviorConfiguration="bookSearchBehavior">
<endpoint
      binding="wsHttpBinding"
      contract="Library.IBookSearch"
      bindingConfiguration="bookSearchBinding">
</endpoint>
</service>
</services>
```

Fig. 5: ***L'Identity Selector per la selezione della card da inviare al web service.***

Dopo aver configurato il servizio, dobbiamo ora configurare il client per consentire l'esecuzione dell'Identity Selector nella fase di contatto del servizio. Aggiungiamo quindi lo stesso blocco binding bookSearchBinding specificato nella parte server, ed in più indichiamo un behavior per recuperare le informazioni del certificato:

```
  <behaviors>
    <behavior name="bookSearchBehavior">
      <clientCredentials>
        <serviceCertificate>
          <authentication
      certificateValidationMode="PeerOrChainTrust"/>
          <defaultCertificate
           findValue="faw2k3"
           storeLocation="LocalMachine"
           storeName="My"
           x509FindType="FindBySubjectName" />
        </serviceCertificate>
      </clientCredentials>
    </behavior>
  </behaviors>
```

L'impostazione del certificateValidationMode su PeerOrChainTrust richiede la presenza del certificato tra le fonti attendibili dell'utente. Ora impostiamo il codice sul client per accedere al servizio e selezioniamo l'identity da utilizzare:

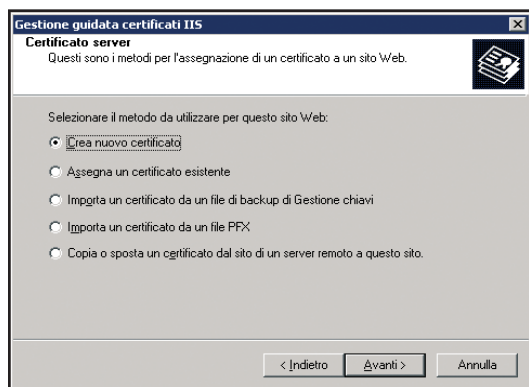
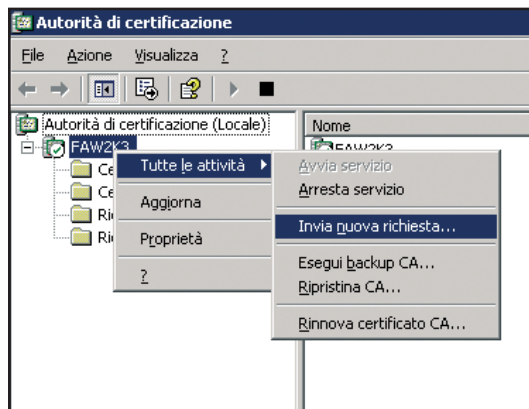
```
  <client>
    <endpoint
        name="BookSearchEndpointConfig"
address="http://faw2k3/ioProgrammo/BookService/B
                        ookService.svc"
        binding="wsHttpBinding"
        contract="Library.IBookSearch"
        bindingConfiguration="bookSearchBinding">
      <identity>
        <certificateReference
          findValue="faw2k3"
          storeLocation="LocalMachine"
          storeName="My"
          x509FindType="FindBySubjectName" />
      </identity>
    </endpoint>
  </client>
```

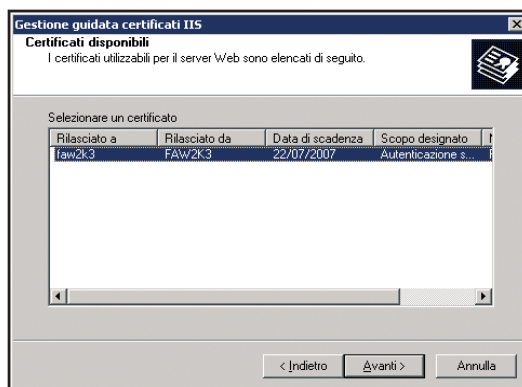
Se è tutto impostato correttamente, quando proviamo ad eseguire l'applicazione dovrebbe com-

parirci l'Identity Selector che ci consente la selezione della card da inviare al web services.

CREARE UN CERTIFICATO

L'articolo descrive come utilizzare un certificato X.509 con Windows CardSpace.
Vediamo come possiamo creare un certificato utilizzando la Certification Authority fornita con Windows Server 2003.

1 Apriamo IIS 6, facciamo click con il tasto destro sul Sito Web predefinito e selezioniamo *Proprietà*. Apriamo la scheda *Protezione directory* e clicchiamo sul tasto *Certificato Server*. Ora dobbiamo richiedere un nuovo certificato selezionando l'opzione *Crea nuovo certificato* ed impostiamo le proprietà del certificato che andiamo a creare. Infine salviamo la richiesta in un file txt.

Passo 1: **Selezioniamo la creazione del nuovo certificato**

Passo 2: **Elaboriamo la richiesta**

2 Apriamo ora il tool *Autorità di certificazione* in Strumenti di amministrazione.
Clicchiamo con il tasto destro sul server visualizzato, selezioniamo *Tutte le attività* e poi *Invia nuova richiesta*.
Apriamo qui il file txt creato al punto 1. Ora nella scheda Richieste in sospeso troveremo la richiesta appena inviata, facciamo clic con il tasto destro sulla richiesta e selezioniamo *Tutte le attività* e poi *Emetti*. Con questa azione abbiamo emesso il certificato che è ora visibile nella scheda Certificati emessi.

Passo 3: **Selezioniamo il certificato creato.**

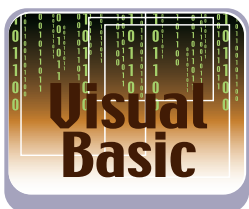
3 Ora riapriamo IIS e selezioniamo le Proprietà del Sito Web predefinito. Ritorniamo nella scheda Protezione directory e clicchiamo sul tasto *Certificato server*.
Selezioniamo ora la scelta *Assegna un certificato esistente* e selezioniamo il certificato appena creato, come in figura. Clicchiamo su *Avanti* fino a terminare il wizard. Ora il certificato è stato associato al server.

CONCLUSIONI

Un sistema innovativo come quello realizzato da WCS impiegherà probabilmente qualche tempo per essere recepito dalla maggior parte delle entità coinvolte. Qualcosa già si sta muovendo e molti progetti open source stanno convergendo verso il sistema unico, sintomo che la strada intrapresa è probabilmente quella giusta. Evitare di fornire, ogni volta che è necessaria una autenticazione, una username ed una password garantisce e limita il pericolo di smarrimento o intercettazione della stesse credenziali. I limiti sono probabilmente quelli legati all'uso di protocolli sicuri come l'SSL, un vincolo che probabilmente impedirà a siti minori di rientrare nel circuito a causa dei costi di certificati. Quello della gestione dell'identità digitale è un problema che non può essere sottovalutato alla luce dell'erogazione di nuovi servizi che sfruttando la tecnologia internet migliorano vistosamente la qualità della vita. Sicuramente con il passare del tempo le società erogatrici dei certificati digitali dovranno adeguare i loro prezzi alla richiesta del mercato

Fabio Cozzolino

DENTRO IL SISTEMA USIAMO LE API

ALCUNE TECNICHE AVANZATE CHE CONSENTONO DI INTERAGIRE DIRETTAMENTE CON LE PARTI PIÙ INTERNE DEL SISTEMA OPERATIVO. ECCO COME RICHIAMARE E UTILIZZARE LE FUNZIONI PIÙ INTERNE PER PERSONALIZZARNE IL COMPORTAMENTO

Le *API* (*Application Programming Interface*) sono i *blocchi* di codice sui quali si costruiscono le applicazioni *Windows-Based*. Le funzioni dell'*API* sono supportate sia dai sistemi operativi a *32-bit* sia da quelli a *64-bit*. Le *API* sono tante e possono essere usate per svariati scopi, per tale ragione sono organizzate in categorie, per esempio la categoria *Timer Functions* contiene l'insieme degli elementi che permettono di definire e gestire dei *Timer*. Come vedremo nel corso dell'articolo sono: *SetTimer*, *KillTimer ecc*. Le *API* sono racchiuse in delle *DLL* (*dynamic-link library*) cioè dei file che contengono procedure, funzioni ecc e che sono caricate e collegate alle applicazioni a *Run_Time*. Il modo migliore per includere nei progetti *Visual Basic* le caratteristiche dell'*API* è utilizzare l'applicazione *APIload.exe*.

Quando le *API* sono utilizzate con particolari tecniche di programmazione, come *CallBack* e *SubClassing*, è possibile estendere ulteriormente le caratteristiche dell'ambiente di programmazione e dei suoi elementi nativi ed avere il controllo di buona parte dell'interazione tra il sistema operativo e l'interfaccia dell'applicazione. Quando si utilizzano questi strumenti, però, bisogna essere precisi, come dei chirurghi, dato che si rischia di mandare in *Crash* il Sistema Operativo.

In questo articolo presenteremo alcuni esempi che mettono in luce le potenzialità delle *API*, del *CallBack* e del *SubClassing*. In particolare per quanto riguarda il *CallBack* presenteremo un esempio che permette di valutare la posizione del cursore del *Mouse* ed il tempo trascorso dall'ultimo avvio del Sistema Operativo. Il *SubClassing*, invece, lo utilizzeremo per ampliare le caratteristiche del menu predefinito delle *Form*, cioè il menu *Drop-Down*. Prima di presentare gli esempi, però, è necessario fare una breve rassegna sulle *API*, sul *CallBack* e il *SubClassing* e sugli argomenti correlati.

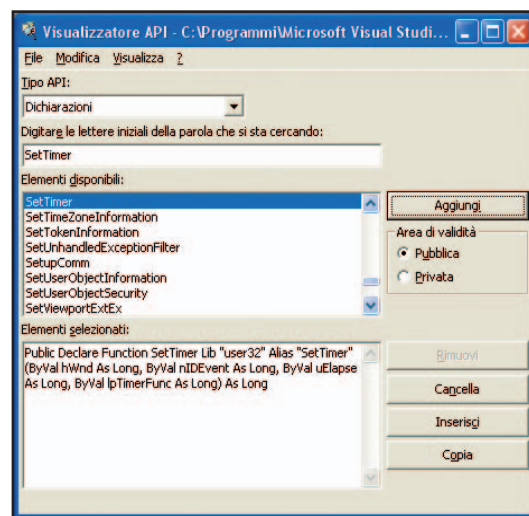
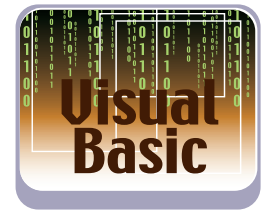



Fig. 1: *L'applicazione APILoad.exe*

CALLBACK

La tecnica di *CallBack*, consiste nel gestire determinati eventi, generati dal sistema operativo, con delle funzioni dette *Window Procedure* o funzioni di *CallBack*. Le funzioni di *CallBack* dal punto di vista del programmatore sono associate a precisi messaggi del sistema e sono implementate in base ad un'interfaccia definita nelle *API*. Le funzioni di *CallBack* sono *Event-Driver*, nel senso che non sono eseguite in modo sequenziale, come le normali funzioni, ma in base al verificarsi di un evento nel sistema operativo. Di solito le funzioni di *CallBack* devono eseguire un'azione ripetutamente, per esempio enumerare *finestra*, *Font*, *Printer ecc.* cioè operazioni associate ad API come *EnumWindows*, *EnumPrinters*, *EnumFontFamilies*, *ecc.* Per questo la tecnica di *CallBack* serve a produrre determinate informazioni richiamando una o più volte una procedura all'interno di un'altra dove l'interazione tra le routine è corredata da un insieme di parametri che consentono di passare alla funzione di *Callback* (quella più interna) i dati prodotti dalla funzione chiamante (quella esterna).

La tecnica di CallBack dunque si basa sull'utilizzo di una funzione con un parametro di tipo puntatore a funzione. Questo parametro consente il passaggio di una funzione da utilizzare per elaborare i dati prodotti dalla funzione chiamante. Questo passaggio da *Visual Basic* è gestito con l'operatore *AddressOf*.

CALLBACK PER TIMER E MOUSE

Presentiamo un esempio che permette di verificare la posizione del *Mouse* e il tempo trascorso dall'avvio del sistema operativo. Visualizzeremo queste informazioni con delle *label* e le aggiorneremo in base ad un *Timer* impostato utilizzando una funzione di *CallBack*. A tal fine utilizzeremo i seguenti elementi dell'*API*: *SetTimer*, *KillTimer*, *GetCursor Pos* e il tipo *PointerApi*. Inoltre, implementeremo la funzione di *CallBack MyTimerProc* definita in base all'interfaccia della *Window Procedure TimerProc* che elabora il messaggio *WM_TIMER*. Iniziamo a descrivere le *API*. La *SetTimer* crea un *Timer* con un particolare intervallo di tempo di scansione cioè di *TimeOut*. Essa ha la seguente sintassi:

```
Public Declare Function SetTimer Lib "user32" _
(ByVal hwnd As Long, ByVal nIDEvent As Long, _
ByVal uElapse As Long, ByVal lpTimerFunc _
As Long) As Long
```

I parametri più importanti della *SetTimer* sono *uElapse*, che serve per specificare la dimensione dell'intervallo di *TimeOut* (esso è equivalente a *Interval* del controllo primitivo *Timer*) e *lpTimerFunc* che rappresenta un puntatore ad una funzione, cioè ad una funzione di *CallBack* richiamata allo scadere del *TimeOut*. La *SetTimer* restituisce un valore intero che rappresenta l'identificativo del *Timer*, questo valore deve essere utilizzato, quando si uccide (*Kill*) il *Timer* con la funzione *KillTimer* che descriveremo tra dopo. Nella *SetTimer* la funzione che sarà specificata con il parametro *lpTimerFunc* è richiamata (direttamente da *Windows*) solo se il suo nome è preceduto dall'operatore *AddressOf*. Quest'ultimo permette il passaggio dell'indirizzo (a 32 bit) della funzione alla routine *API*. La *KillTimer* ha la seguente sintassi.

```
Public Declare Function KillTimer Lib "user32" _
(ByVal hwnd As Long, ByVal nIDEvent As Long) _
As Long
```

Dove *nIDEvent* è l'identificatore del *Timer* da uccidere. Per leggere la posizione del *Mouse* invece utilizziamo i seguenti elementi.

```
Public Type POINTAPI
        x As Long
        y As Long
End Type
Public Declare Function GetCursorPos _
Lib "user32" (lpPoint As POINTAPI) As Long
```

La funzione *GetCursorPos*, che appartiene alla *DLL user32.dll*, ha un solo parametro del tipo *POINTAPI*. Quest'ultimo è una struttura che aggrega le coppie di coordinate (*X,Y*) che specificano la posizione del Mouse sul *video*.
Ora possiamo implementare il programma che permette di valutare la posizione del *Mouse* ed il tempo trascorso dall'avvio del sistema.

1 Create un progetto con un modulo *Bas* ed una *Form* (nominata *FRMCallBack*), su quest'ultima predisponete due Label nominate *LabTime* e *LabMouse* che utilizzeremo per mostrare le informazioni generate nella funzione di *CallBack*. Nel modulo, invece, dovete inserire: le dichiarazioni degli elementi dell'*API* descritte prima; la dichiarazione della variabile che identifica il *Timer* (*NumTim*) e la funzione di *CallBack MyTimerProc*.

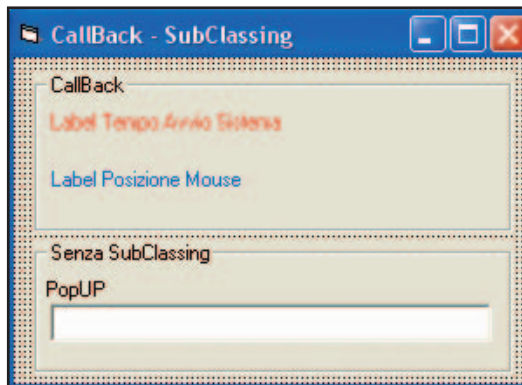
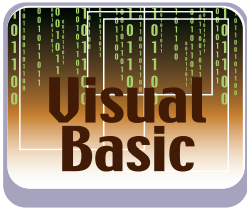



Fig. 2: **La Form in fase di progettazione**

```
Public NumTim As Long
Sub MyTimerProc(ByVal hwnd As Long, ByVal uMsg
                                          As Long, _
ByVal idevent As Long, ByVal systime As Long)
 Dim PosMouse As POINTAPI
 GetCursorPos PosMouse
 FRMCallBack.LabMouse = "Posizione Mouse: " + _
  " X=" + Str(PosMouse.x) + " Y=" +
                                   Str(PosMouse.y)
 DoEvents
 FRMCallBack.LabTime = "Minuti Trascorsi dall'avvio"
                                                 _
 & " del sistema: " + CStr((Round(systime / 60000,
                                             2)))
End Sub
```

NumTim è l'identificatore del *Timer* che, come vedremo avanti, creiamo nella *Form_Load* con la

SetTimer. Nella *MyTimerProc LabMouse* e *LabTime* sono le *label* inserite sulla *FRMCallBack*, mentre il parametro *systime* contiene il numero di millisecondi trascorsi dall'avvio del sistema operativo, con *Round(systime / 60000, 2)* lo convertiamo in minuti. *PosMouse* contiene le coordinate del cursore del *Mouse*; *PosMouse* è aggiornato anche quando il *Mouse* non si trova sulla *Form*. Per capire meglio la *MyTimerProc* vi conviene controllare, nel punto successivo, il codice che l'attiva.

2 Nella *Form_Load* inseriamo il codice che imposta il *Timer*.

```
Private Sub Form_Load()
 NumTim = SetTimer(Me.hwnd, 0, 100, _
 AddressOf MyTimerProc)
End Sub
```

La *SetTimer* imposta il *Timer*, con *TimeOut=100 millisecondi*, e gli associa la funzione di *CallBack MyTimerProc*. Per disabilitare il *Timer* usiamo la funzione *KillTimer* con l'identificatore del *Timer* – *NumTim* - come argomento. Questo lo dobbiamo fare nella *Form_Unload*.

```
Private Sub Form_Unload(Cancel As Integer)
 KillTimer Me.hwnd, NumTim
End Sub
```

SUBCLASSING

La tecnica del *Subclassing* consiste nella creazione di *classi secondarie* con lo scopo di intercettare e processare i messaggi spediti dal sistema operativo alle finestre attive, cioè agli oggetti dotati di un *handle*, che implementano l'interfaccia dell'applicazione. In parole povere, quando il sistema operativo deve comunicare ad un programma che l'utente ha fatto una certa azione, sulla tastiera o con il *Mouse*, invia un messaggio agli oggetti della *form* attiva, questo è elaborato dalla funzione di *CallBack* predefinita (che implementa le azioni da eseguire per rispondere all'utente) associata a quell'oggetto. Dal punto di vista del programmatore lo scopo del *SubClassing* è agganciare (*Hook*) il messaggio per bypassare la *CallBack* di *default* ed eseguire una funzione di *CallBack* personalizzata. Per esempio, quando l'utente clicca con il tasto destro del *Mouse* su un *TextBox* risponde la *CallBack* predefinita che visualizza il menu di *PopUp* classico con i comandi per gestire i dati presenti nel *TextBox*, se creassimo una classe secondaria (*SubClassing*) del *TextBox* potremmo intercettare quest'azione e mostrare un altro menu. Di seguito descriviamo come fare il *SubClassing* del menu *Drop-Down* di una *Form*. Prima di procedere ribadiamo alcuni concetti. Ogni finestra è individuata attraverso il suo *handle*, che è un'espressione numerica, un *long*. Gli oggetti con finestra sono tutti gli oggetti dell'interfaccia, per capirci anche i *TextBox* ed i *Menu*. Essi comunicano con il sistema operativo con un protocollo a messaggi. Ad esempio quando l'utente ridimensiona una finestra, il sistema operativo invia un messaggio all'applicazione per comunicare ciò che sta avvenendo, facendo in modo che il programma reagisca di conseguenza. In altre parole quando *Windows* deve inviare un certo messaggio ad un'applicazione non fa altro che chiamare una funzione, detta *Window Procedure*, alla quale passa come argomento il messaggio da comunicare. Il *Subclassing* dunque consiste nel definire delle *Window Procedure* personalizzate in sostituzione di quelle predefinite. Naturalmente questo non deve essere fatto per tutti i messaggi che *Windows* può inviare ad un dato oggetto!

IL SUBCLASSING DI UN MENU

In questo paragrafo presentiamo la tecnica di *SubClassing* servendoci di un esempio molto interessante: mostreremo come personalizzare il menu *Drop-Down* di una *Form*, cioè il menu che si attiva

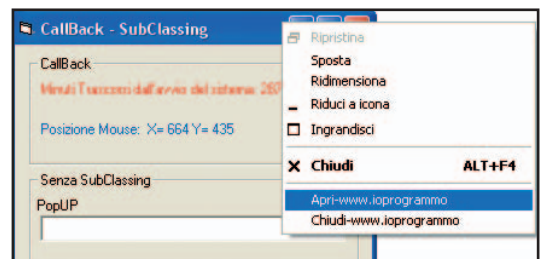
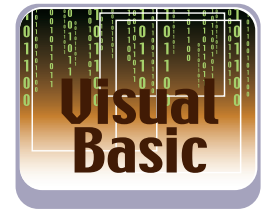



Fig. 3: *Il menu Drop-Down modificatto*

quando si clicca con il tasto destro del *Mouse* sul suo bordo. Per far ciò ci serviremo di diverse funzioni *API*, iniziamo a descrivere le più importanti. *SetWindowLong*, della libreria *API "user32"*, consente di sostituire una procedura di *CallBack* di sistema con quella definita dall'utente (in pratica crea la Sotto Classe), in particolare va a sostituire, nella tabella dati associata a ciascuna finestra, l'indirizzo della procedura di *default* con quello della nostra procedura. La sintassi è la seguente:

```
Public Declare Function SetWindowLong Lib "user32"
 Alias _
"SetWindowLongA" (ByVal hwnd As Long, _
ByVal nIndex As Long, ByVal dwNewLong As Long)
As Long
```

Descriviamo brevemente i parametri di questa

funzione. *Hwnd* è l'*handle* della finestra, *NIndex* è l'indice della tabella dati interna in cui vogliamo memorizzare il nuovo valore, *DwNewLong* è il nuovo valore che si vuole memorizzare, puntato da *NIndex*. Notate che *SetWindowLong* restituisce il valore che era presente in quella locazione di memoria, valore che deve essere conservato e ripristinato, al termine dell'esecuzione della nostra procedura di *CallBack*, per evitare errori sul sistema. La funzione *API* che si occupa della chiamata alla *Window Procedure*, effettuando l'invio del messaggio, è la *CallWindowProc* che bisogna dichiarare a livello di modulo insieme alla *SetWindowLong*.

```
Public Declare Function CallWindowProc Lib "user32" Alias _
"CallWindowProcA" (ByVal lpPrevWndFunc As Long, _
    ByVal hwnd As Long, ByVal Msg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long
```

I parametri di questa funzione sono: *LpPrevWndFunc* che è l'indirizzo della procedura originale *Hwnd* , *Msg* , *wParam*, *lParam* che sono gli argomenti di ritorno della procedura personalizzata.
Altre due funzioni che dobbiamo utilizzare nel nostro esempio sono *AppendMenu* e *GetSystemMenu* che hanno la seguente sintassi.

```
Public Declare Function AppendMenu Lib "user32" Alias "AppendMenuA" _
(ByVal hMenu As Long, _
ByVal wFlags As Long, _
ByVal wIDNewItem As Long, _
ByVal lpNewItem As String) As Long
Public Declare Function GetSystemMenu Lib "user32" _
(ByVal hwnd As Long, _
ByVal bRevert As Long) As Long
```

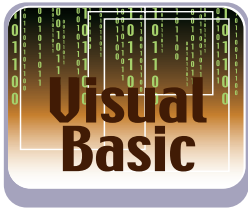
Come vedremo, nel codice dell'esempio, l'AppendMenu permette di appendere una nuova voce di menu alla fine di uno specifico Drop-Down menu. I parametri hanno il seguente significato: hMenu è l'identificatore del menu da modificare; uFlags è un flag che specifica lo stile ed il comportamento del menu; *uIDNewItem* è l'identificatore del nuovo menu e *lpNewItem* specifica il contesto del nuovo menu, esso dipende dal parametro *uFlags*.
GetSystemMenu, invece, permette ad un'applicazione di accedere ad un menu di *Windows*, cioè ad un *system menu*, per modificarlo o copiarlo. I parametri sono i seguenti: *hWnd* che è l'*handle* della finestra proprietaria del Menu; mentre *bRevert* è un *boolean* che specifica l'azione da fare, se è *False* la funzione ritorna l'*handle* della copia del *Window Menu* correntemente in uso altrimenti restituisce *Null*.

Vediamo per punti come implementare il progetto che personalizza il menu di una *Form*. In particolare al menu della *Form* aggiungiamo i comandi che permettono di aprire e chiudere un sito *Web* tramite un'istanza di *Internet Explorer*.

1 Create un progetto con una *Form* ed un modulo *Bas*. In quest'ultimo inserite la definizione delle procedure illustrate in precedenza e le seguenti variabili e costanti il cui significato in parte è specificato come commento al codice.

```
Public Const WM_SYSCOMMAND = &H112
'Messaggio ricevuto dalla finestra
Public Const MF_SEPARATOR = &H800&
'Linea che divide le voci di menu
Public Const MF_STRING = &H0&
'Specifica che la voce di menu è una stringa
Public Const GWL_WNDPROC = (-4)
'Indice per creare una sotto classe con
SetWindowLong
Public VecchiaProc As Long
'Indirizzo Window Procedure di Default
Public Const IDM_MenuApri As Long = 1010
'Identificatore voce di menu Apri
Public Const IDM_MenuChiudi As Long = 1020
'Identificatore voce di menu chiudi
Public IEApp As Object
'Oggetto Internet Explorer
```

PUNTATORI

I puntatori sono delle variabili che non contengono dati ma indirizzi di altre variabili. Ci sono dei puntatori particolari che invece di contenere l'indirizzo di una variabile, contengono l'identificatore di una procedura, questi puntatori sono detti Puntatori a Funzioni. In Visual Basic i puntatori a variabili vengono usati, ad esempio, quando si gestiscono gli oggetti di classi definite dagli utenti, i puntatori a funzione, invece, vengono usati per implementare particolari tecniche come il Callback. L'utilizzo dei puntatori a funzioni è reso possibile dall'operatore AddressOf, disponibile dalla versione 5 del linguaggio. Quando si utilizzano i puntatori a funzione è facile perdere il controllo dell'ambiente di sviluppo, per questo vi consigliamo di salvare spesso il progetto e di fare delle copie di backup. Inoltre segnaliamo che se si esegue una funzione di Callback in modalità interruzione si possono riscontrare delle disfunzioni con inconvenienti che possono arrivare fino ad errori di tipo General Protection Fault (GPF). Infine ricordate che i puntatori a funzioni Basic to Basic non sono supportati, sono supportati solo puntatori da Visual Basic ad una funzione DLL.

2 Nel Modulo, inoltre, vanno inserite la funzione che definisce le due voci di menu e l'appende al menu della *Form* e la *Window Procedure* che implementa il *SubClassing*. Cioè le procedure *SubMenuForm* e la *GestMenuForm*. Quest'ultima è quella di *CallBack* ed è invocata nella *SubMenuForm*.

```
Public Function SubMenuForm(param As Form)
 Dim IDSysMenu As Long, ValRet As Long
 IDSysMenu = GetSystemMenu(param.hwnd, 0&)
 ValRet = AppendMenu(IDSysMenu,
MF_SEPARATOR, _
 0&, vbNullString)
 ValRet = AppendMenu(IDSysMenu, MF_STRING, _
 IDM_MenuApri, "Apri-www.ioprogrammo")
ValRet = AppendMenu(IDSysMenu, MF_STRING, _
IDM_MenuChiudi, "Chiudi-www.ioprogrammo")
 param.Show
 VecchiaProc = SetWindowLong(param.hwnd, _
 GWL_WNDPROC, AddressOf GestMenuForm)
End Function
```

Nella *SubMenuForm* prima è individuato, con la *GetSystemMenu*, il menu da modificare e poi sono aggiunte le voci di menu che permettono di aprire e chiudere un sito. La funzione *GestMenuForm*, come anticipato, è richiamata con la *SetWindowLong*.

```
Public Function GestMenuForm(ByVal hwnd As
Long, _
ByVal iMsg As Long, ByVal wParam As Long, _
ByVal lParam As Long) As Long
If iMsg = WM_SYSCOMMAND Then
Select Case wParam
Case IDM_MenuApri
   ApriSito
  Exit Function
 Case IDM_MenuChiudi
   ChiudiSito
  Exit Function
 End Select
End If
GestMenuForm = CallWindowProc(VecchiaProc, _
hwnd, iMsg, wParam, lParam)
End Function
```

La *GestMenuForm* ha i seguenti parametri: *hwnd*, cioè l'*handle* alla finestra a cui è diretto il messaggio. iMsg, l'identificatore del tipo di messaggio; infine *wParam* e *LParam* che sono delle variabili il cui valore dipende dal messaggio. La suddetta procedura quando è invocata, sul menu della *Form*, visualizza le nuove voci di menu senza alterare quelle preesistenti, infatti, tutti i messaggi non gestiti dalla nostra procedura sono restituiti a *Windows* tramite la funzione *CallWindowProc*, cioè sono passati alla Window Procedure di default. S'intuisce che il messaggio *WM_SYSCOMMAND* è ricevuto dalla *Form* quando l'utente seleziona una voce di menu.

3 Le procedure invocate, quando si selezionano le nuove voci di menu sono le seguenti.

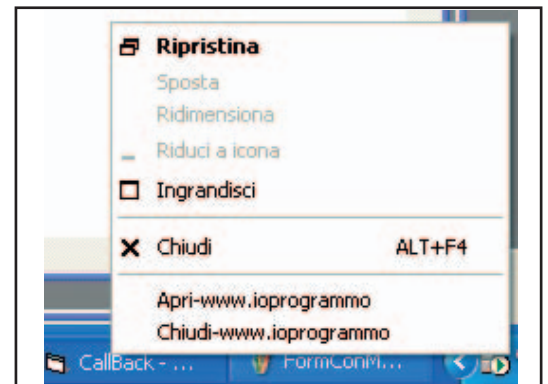



Fig. 4: Il menu Drop-Down quando la Form è iconizzata.

```
Public Sub ApriSito()
 Set IEApp = CreateObject _
 ("InternetExplorer.Application")
 IEApp.Visible = True
 IEApp.Navigate "http://www.ioprogrammo.it"
 IEApp.Height = 500
 IEApp.Width = 500
 IEApp.MenuBar = False
 IEApp.Toolbar = False
End Sub


Public Sub ChiudiSito()
 On Error Resume Next
 IEApp.Visible = False
 Set IEApp = Nothing
End Sub
```

Anch'esse vanno inserite nel modulo del progetto.

L'OPERATORE ADDRESSOF

L'operatore AddressOf permette il passaggio dell'indirizzo (a 32 bit) di una funzione ad una routine API. La sintassi di AddressOf è la seguente:
AddressOf nomeroutine
Dove Nomeroutine è il nome della routine passata all'API.
La funzione richiamata tramite AddressOf deve essere inserita in un modulo (.Bas), quindi non può essere inserita in una classe o in una form; inoltre deve trovarsi nello stesso progetto della routine chiamante.
Per esempio, per contare il numero di finestra aperte, si può scrivere la seguente istruzione.

```
n=EnumWindows(Addressof
                    MiaProcedura,0)
```

Ricordiamo che l'API EnumWindows non fa altro che elencare le windows aperte del sistema. Essa accetta due parametri: il primo serve per specificare l'indirizzo di una funzione di Callback (MiaProcedura, che dovete implementare!) l'altro invece è un parametro che consente di distinguere le varie chiamate della procedura (di solito s'imposta su zero).

4 Nella *Form*, per attivare il *SubClassing* e per disattivarlo basta predisporre le seguenti linee di codice.

```
Private Sub Form_Load()
  SubMenuForm Me
End Sub

Private Sub Form_Unload(Cancel As Integer)
 SetWindowLong Me.hwnd, GWL_WNDPROC, _
 VecchiaProc
End Sub
```

Di seguito presentiamo un metodo alternativo per modificare i *Menu* di un *TextBox*. In particolare non useremo *SubClassing* e funzioni *API*, ma soltanto il metodo *PopupMenu* della *Form*.

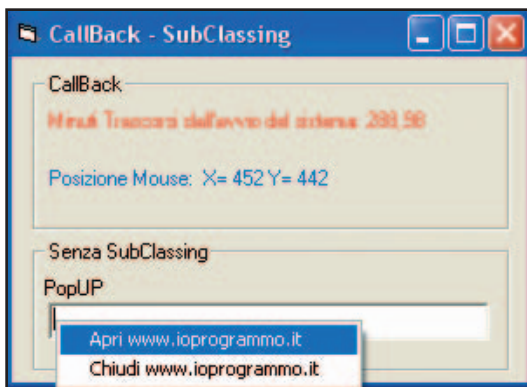



***Fig. 5:** Il menu di un TextBox personalizzato*

1 Create un nuovo progetto e sulla *Form* predisponete un *TextBox* – nominato *TxtPopUp* - e un *Menu PopUP* – la cui procedura principale è nominata *mnuioprogrammo*. Nel Menu prevedete i *menu Apri Sito* e *Chiudi Sito*. Il primo invoca la procedura *ApriSito* il secondo la *ChiudiSito* (per la loro implementazione si controllino gli esempi precedenti).

2 Per bypassare il *Menu* di *Default* dei *TextBox* e mostrare il nostro, basta predisporre il seguente codice nella *TxtPopUp_MouseDown*.

```
Private Sub TxtPopUp_MouseDown(Button As
Integer, -_
 Shift As Integer, x As Single, y As Single)
 If Button = vbRightButton Then
  TxtPopUp.Enabled = False
  DoEvents
  TxtPopUp.Enabled = True
  PopupMenu mnuioprogrammo
 End If
End Sub
```

3 Le procedure associate alle voci di menu, invece, sono le seguenti

```
Private Sub mnuapri_Click()
 ApriSito
End Sub

Private Sub mnuchiudi_Click()
 ChiudiSito
End Sub
```

Ricordiamo che per il *PopUpMenu* è necessario un menu a tendina con almeno una voce di menu che ha un sottomenu, come mostrato in figura 6.

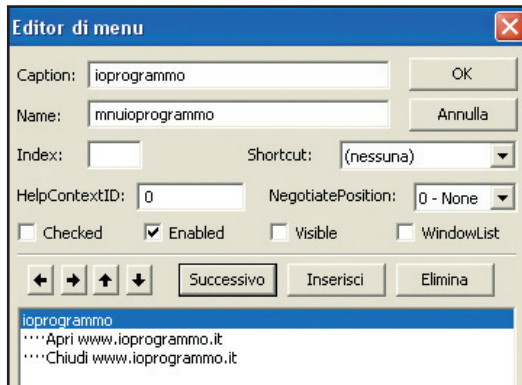



***Fig. 6:** Il Menu PopUp nell'Editor di Menu*

CONCLUSIONI

I metodi qui presentati lavorano al cuore del sistema. Il loro indirizzo non è semplice e immediato, ma acquisendo una buona padronanza si può davvero modificare il comportamento del sistema a proprio gradimento.

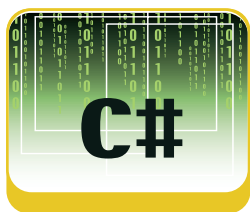
Massimo Autiero

VISUALIZZATORE API

Il Visualizzatore API - Apiload.exe - fornito con Visual Basic è selezionabile sia come singola applicazione (da Start - Visual Studio 6/ Strumenti di Visual Studio/ API Text Viewer) che come Add-in di Visual Basic (Menu: Aggiunte/Gestione Aggiunte). Nel nostro caso è necessario scegliere la seconda opzione dato che permette di copiare direttamente, nella parte dichiarativa di un modulo, gli elementi scelti. Di seguito descriviamo come procedere per fare ciò.

1. Create un nuovo progetto Visual Basic.
2. Inserite Visualizzatore API da gestione aggiunte; quest'azione fa comparire, nel menù aggiunte, la voce Visualizzatore API.
3. Dal menu aggiunte selezionate il Visualizzatore.
4. Dal menu file del Visualizzatore API selezionate il file di testo (menu: file/carica file di testo) o di database (carica file di database) che contiene le definizioni dell'API, per esempio Win32api.txt che contiene la maggior parte delle costanti e delle routine API utilizzabili in Visual Basic.
Dopo il caricamento del file, il visualizzatore è abilitato e presenta i seguenti elementi:
- un combobox, tipo api,
- un textbox, per impostare delle ricerche,
- le listbox Elementi disponibili ed Elementi selezionati,
- alcuni pulsanti.

UNA SEGRETARIA PER LIVE MESSENGER

IL NUOVO MICROSOFT LIVE MESSENGER PORTA CON SÈ SVARIATE INNOVAZIONI. UNA TRA TUTTE È LA FACILITÀ CON CUI È POSSIBILE ESTENDERLO. ATTRAVERSO SEMPLICI CLASSI .NET SI POSSONO AGGIUNGERE FUNZIONALITÀ IMPENSABILI FINO ALLA VERSIONE PRECEDENTE

Con Live Messenger, Microsoft inaugura l'uscita della linea di prodotti Live con i quali vuole "conquistare" Internet, eterno terreno di scontro tra le big del settore.
Live Messenger rompe il classico versioning, seguito dal big di Redmond fin'ora, per il suo client di Instant Messaging.
Questa rottura sottolinea anche un nuovo set di feature impensabili fino alla versione precedente: una di queste, che andremo ad analizzare in questo articolo, è la possibilità di sviluppare AddIn attraverso semplici classi ed un qualsiasi linguaggio .NET 2.0-compliant, con l'ausilio dell'immancabile Visual Studio 2005.
Il primo passo da fare è abilitare la feature degli AddIn (ancora nascosta) in Windows Live Messenger come descritto dal box laterale.
Queste brevi operazioni preliminari sono necessarie perché, ad oggi, non è ancora disponibile un SDK per la piattaforma Live: questa metterà a disposizione Template già configurati per lo sviluppo di AddIn sia con VS2005 sia con Visual C# Express.

IMPLEMENTAZIONE DI UN ADDIN DI BASE

Affinchè si Live Messenger possa caricare il nostro AddIn, questo deve seguire 3 semplici regole:

1. Il nome della DLL (ovvero dell'Assembly) dovrà avere lo stesso nome della classe principale (nel formato Namespace.NomeClasse).
2. Il progetto deve referenziare la DLL MessengerClient.dll .
3. La classe principale dovrà implementare l'interfaccia IMessengerAddIn (ovvero il metodo Initialize, l'unico dell'interfaccia).

Il primo passo da fare è, dunque, creare un progetto di tipo DLL (Windows Class Library) assegnando un nome consono e significativo.

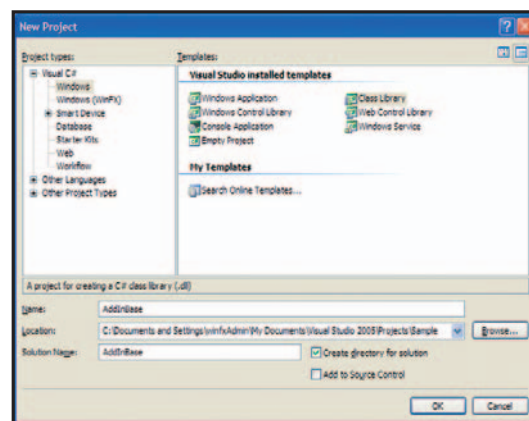



Fig. 1: ***Creazione di un progetto Class Library***

Come si potrà notare, Visual Studio creerà un file di classe (Class1.cs) con il namespace uguale al nome del progetto (AddInBase); sarà nostro compito, dunque, cambiare anche il nome della classe appena creata (in Figura 1 è stato cambiato in MioAddIn).
Il nome della DLL, quando compileremo il progetto, dovrà essere: AddInBase.MioAddIn.dll, diverso da quello che assegnerebbe VS2005 di default (AddInBase).
Per cambiare quesst̀o comportamento è sufficiente aprire le proprità del progetto e cambiare il valore del campo Nome Assembly da AddInBase in AddInBase.MioAddIn (Figura 2).

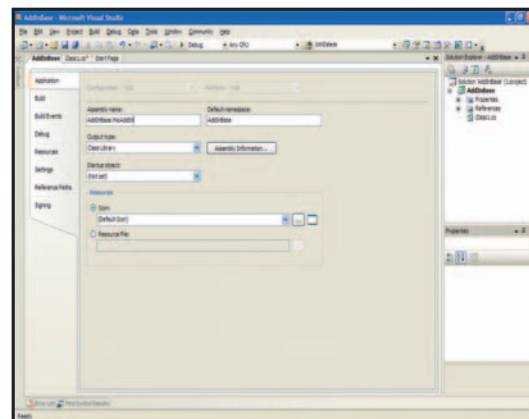



Fig. 2: ***Cambio del nome di un assembly***

Continuando nel soddisfare i requisiti affinchè la nostra DLL possa fregiarsi del nome AddIn, aggiungiamo il riferimento della DLL MessengerClient.dll al nostro progetto.
Tale DLL si trova nella cartella di installazione del Live Messenger: tipicamente C:\Programmi\MSN Messenger (Figura 3).

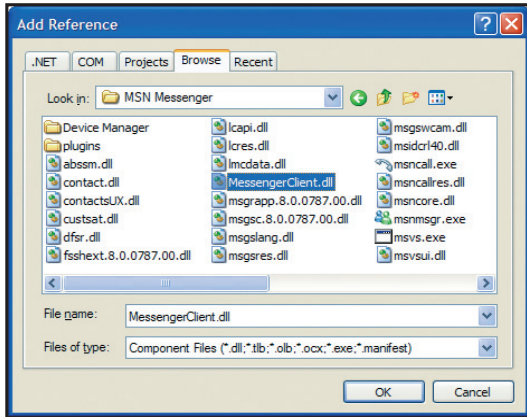



Fig. 3: *Aggiungere il riferimento a Live Messenger.*

Finalmente possiamo addentrarci nel codice dell'AddIn, dapprima soddisfando l'ultimo requisito, e poi descrivendo tutto l'Object Model a cui possiamo finalmente accedere.
Modifichiamo la classe Class1.cs del progetto, aggiungendo dapprima l'istruzione using e poi l'implementazione dell'interfaccia *IMessengerAddIn*.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Messenger;
namespace AddInBase
{
    public class MioAddIn : IMessengerAddIn
    {
        protected MessengerClient msn;
        #region IMessengerAddIn Members
        public void Initialize(MessengerClient messenger)
        {
            this.msn = messenger;
            this.msn.AddInProperties.Creator = "Igor Antonacci";
            this.msn.AddInProperties.Description = "Il mio primo AddIn";
            this.msn.AddInProperties.FriendlyName = "Primo AddIn";
this.msn.AddInProperties.PersonalStatusMessage = "Ciao dal primo AddIn";
        }
        #endregion
    }
}
```

Il metodo Initialize è l'EntryPoint dell'AddIn: qui ci viene passato il riferimento all'oggetto messenger che ci permetterà di interagire con il motore di Live Messenger stesso.
Attraverso la proprietà AddInProperties è possibile impostare i valori relativi all'autore, alla descrizione e persino di alcune proprietà normalmente modificabili dall'utente come il messaggio personale (PersonalStatusMessage).
Con questo semplice listato abbiamo realizzato il nostro primo AddIn il cui scopo è modificare il messaggio personale dell'utente mostrando "Ciao dal primo AddIn".
Ultimo passo affinchè Live Messenger veda il nostro AddIn è, innanzitutto compilare il progetto e successivamente copiare il file AddInBase.MioAddIn.dll nella cartella di installazione del Messenger.
A questo passo va aggiunto quello del caricamento vero e proprio dell'AddIn: dal menu
Strumeti ‡ Opzioni ‡ AddIn, clickare sul bottone "Aggiungi a Messenger"; cisarà permesso di scegliere la DLL appena copiata.
Finalmente il nostro AddIn è stato aggiunto tra quelli che Live Messenger riconosce.

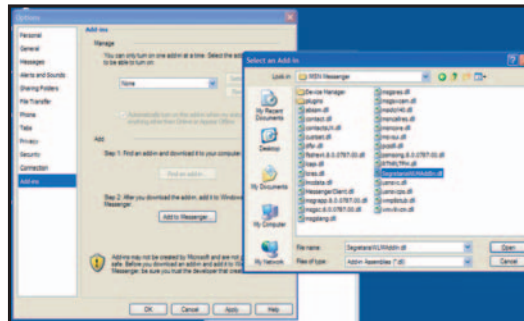



Fig. 4: *Aggiungere l'AddIn a Live Messenger*

Selezionando il checkbox "Abilita automaticamente l'AddIn quando il mio stato è diverso da Online e Compari Offline" l'addin si attiverà SOLO quando il vostro stato è diverso da Online e Appari Offline.
Nel caso vogliate attivarlo manualmente, sarà sufficiente selezionare "Abilita Primo AddIn" dalla finestra di selezione dello stato (Figura 5).

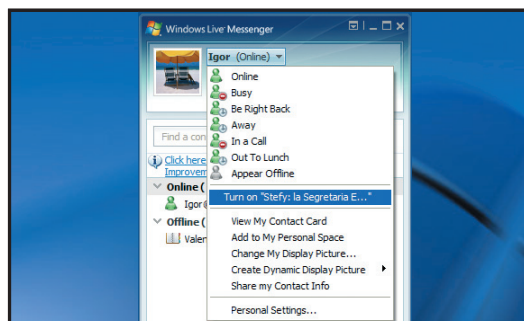
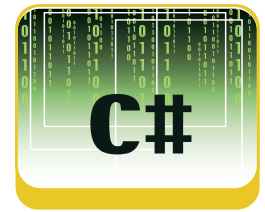
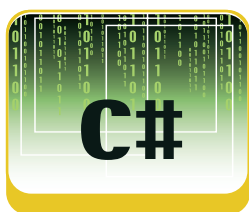



Fig. 5: *Abilitazione manuale dell'AddIn*

WEB REFERENCE

E' possibile trovare materiale a disposizione all'indirizzo:
http://msdn.microsoft.com/library/en-us/dnlive/html/messengeraddin_sdk.asp?frame=true .

STEFY: LA SEGRETARIA ELETTRONICA

Provando ad aumentare un po' la complessità del progetto, possiamo fare in modo che, il nostro AddIn (d'ora in poi semplicemente Stefy) risponda automaticamente alle domande poste dai nostri contatti, magari facendogli credere che ci siamo ancora noi dietro la tastiera o, più semplicemente, per farci lasciare un messaggio che leggeremo al ritorno. Per fare ciò, ci avvaliamo degli eventi che l'oggetto MessengerClient ci mette a disposizione: sottoscrivendo l'evento OnIncomingText, possiamo intercettare il messaggio in arrivo e, quindi rispondere prorio al contatto che ci ha scritto.

```
this.msn.IncomingTextMessage += new
        EventHandler<IncomingTextMessageEventArgs>
                (messenger_IncomingTextMessage);
```

All'interno del gestore dell'evento (*messenger_IncomingTextMessage*), è possibile recuperare molte informazioni passateci sia attraverso l'oggetto sender che tramite l'oggetto e (di tipo *IncomingTextMessageEventArgs*).

```
    void messenger_IncomingTextMessage(object
            sender, IncomingTextMessageEventArgs e)
    {
if (msn.AddInProperties.Status != UserStatus.Offline)
        {
            string domanda = e.TextMessage;
            string risposta = string.Empty;
            if(domanda == "Ciao") {
                risposta = "Ciao " + e.UserForm;
            }
            msn.SendTextMessage(risposta,
                                        e.UserFrom);
        }
    }
```

Tra le tante proprietà spicca sicuramente msn.AddInProperties.Status che ci permette di sapere in che stato è il Messenger al momento dell'arrivo del messaggio. L'oggetto e (argomento dell'evento) porta con se sia il messaggio in arrivo (e.TextMessage), che il riferimento all'utente che l'ha inviato (e.UserForm). Un evento utile a cui Stefy si sottoscrive è quello relativo all'apertura del Form delle impostazioni:

```
this.msn.ShowOptionsDialog += new
        EventHandler(messenger_ShowOptionsDialog);
```

Tale evento viene sollevato nel momento in cui si clicka sul bottone "Impostazioni" dal menu *AddIn di Messenger*. Attraverso questa finestra, possiamo lasciare all'utente la possibilità di specificare alcune opzioni. Nel nostro caso, permettiamo all'Utente di inserire domande a cui Stefy dovrà rispondere quando ci allontaneremo dal PC.

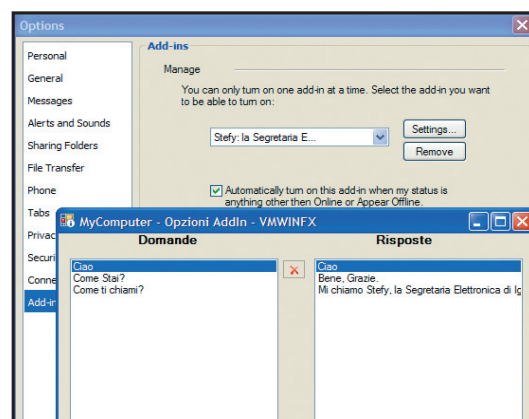



Fig. 6: ***Configurazione dell'AddIn***

Degno di nota è la feature che permette agli AddIn di salvare il proprio stato in una stringa. Stefy si avvale di una classe di servizio (PlugInState) che si serializza in XML: qui vengono memorizzate le domande e le risposte inserite dall'utente con l'ausilio di due Liste sincronizzate. All'uscita dal Form delle Impostazioni, la stringa risultante dalla serializzazione, viene passata alla proprietà *SavedState* dell'oggetto *Messanger Client*.

```
this.msn.SavedState = PlugInState.Save();
```

CONCLUSIONI

La nuova feature di Live Messenger merita di essere esplorata ancora a lungo, magari gestendo gli altri eventi messi a disposizione dall'Object Model. Possibili sviluppi potrebbero essere quelli di aggiungere un motore di Intelligenza Articiale a Stefy così da sembrare un po' più "umana" ed un po' meno "Elettronica".

Igor Antonacci

L'AUTORE

IGOR ANTONACCI

Lavora da circa 8 anni con tecnologie Microsoft, passando da COM e COM+ ed ASP fino all'ultima versione del .NET Framework. Attualmente lavora nel reparto Informatica di una Multinazionale del settore HealthCare. È possibile leggere il suo blog all'indirizzo http://blogs.ugidotnet.org/dotnethoughts . Gli si può scrivere all'indirizzo igor.antonacci@gmail.com .

ABILITARE GLI ADDIN IN LIVE MESSENGER

Gli AddIn sono, purtroppo, disabilitati di default.
Per abilitarli è sufficiente seguire questi brevi passi (oppure eseguire il file EnableAddIn.reg allegato al codice):

1. Chiudere Live Messenger.
2. Dal menu Start, scegliere Esegui e digitare: regedit seguìto dal tasto invio.
3. Sfogliare l'albero del registro seguendo il seguente percorso: HKEY_CURRENT_USER \ Software \ Microsoft \ MSNMessenger
4. Creare un nuovo valore di tipo DWORD (click destro del mouse sul nodo MSNMessenger): AddInFeatureEnabled ed assegnargli il valore 1
5. Al riavvio di Messenger, dopo aver fatto logon, nel menu Strumenti ‡ Opzioni, apparirà una nuova voce: AddIn.

UN TEMPLATE PER VISUAL STUDIO 2005

ECCO COME CREARE LO "SCHELETRO" DI UN'APPLICAZIONE DA RIUTILIZZARE PER TUTTI I PROGETTI SIMILI CHE SVILUPPERAI IN FUTURO. TI SPIEGHIAMO COME PUOI ESTENDERE IL TUO IDE PREFERITO GRAZIE AGLI STARTER KIT

Una delle novità introdotte da Visual Studio 2005 è la possibilità di creare e condividere gli Starter kit. Uno starter kit è un progetto già pronto per l'uso che contiene la documentazione dettagliata che chiarisce i vari componenti del progetto e come interagiscono tra loro.
In pratica, uno starter kit, può essere considerato come un modello che permette di acquisire familiarità con gli strumenti offerti da VB ed imparare tecniche di programmazione avanzate che permettono di generare, senza dover scrivere codice, un programma completamente funzionante. Uno starter kit può essere utilizzato anche come punto di partenza per lo sviluppo di una applicazione, poiché può contenere suggerimenti, codice di esempio e tutta la documentazione necessaria a renderne molto semplice la personalizzazione e l'ampliamento. È inoltre possibile condividere i propri progetti con altri utenti oppure pubblicarli sul Web.

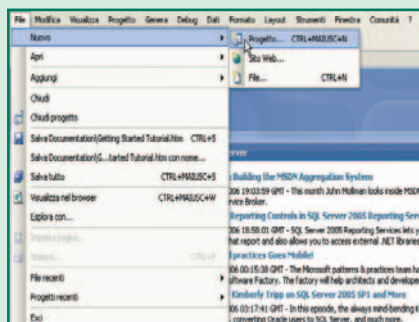
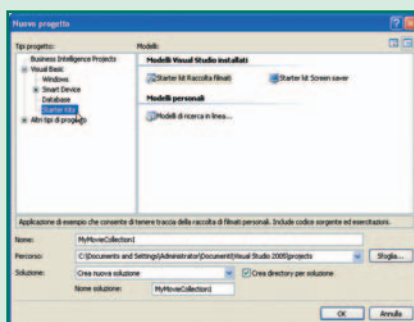
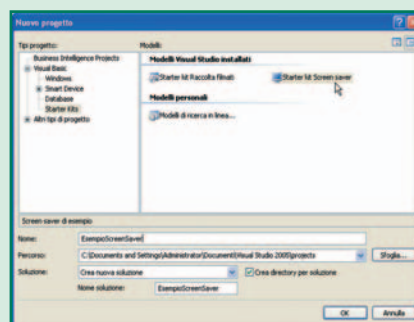
GLI STARTER KIT DISPONIBILI IN VB 2005

In Visual Basic .Net 2005, sono disponibili due starter kit.

- Starter kit: Raccolta filmati
- Starter kit: Screen saver

Lo starter kit Raccolta filmati permette di raccogliere e classificare i propri filmati nonché la ricerca di informazioni in linea. Questo starter kit è un buon esempio di utilizzo di VB ed Sql Server e può essere personalizzato e modificato per raccogliere qualsiasi altro tipo di informazione (file musicali, e-book, ecc)
Dopo aver generato un nuovo progetto dallo starter kit Raccolta filmati, possiamo utilizzarlo per:

- Inserire le informazioni necessarie a descrivere la propria raccolta filmati.
- Archiviare e salvare le informazioni inserite, in un database.

CREARE UN NUOVO PROGETTO DA UNO STARTER KIT

1 Selezioniamo la voce di menu File➞Nuovo Progetto, in questo modo verrà visualizzata la finistra di dialogo: Nuovo progetto che ci consentirà di scegliere lo scheletro da cui iniziare. Anche i template vengono inizializzati da uno starter kit

2 Nella finestra di dialogo Nuovo progetto, dobbiamo selezionare la voce Starter Kits nella parte di sinistra della finestra, che contiene i tipi di progetto. Così facendo, nella parte destra, vengono visualizzati gli starter kits disponibili (Raccolta filmati e Screen saver)

3 A questo punto, non resta che selezionare lo starter kit desiderato nell'elenco di destra: Modelli Visual Studio installati, e cliccare su OK. Sempre in questa finestra, è possibile modificare il nome dell'applicazione dalla casella Nome oppure il percorso di memorizzazione

- Cercare informazioni online sui filmati.

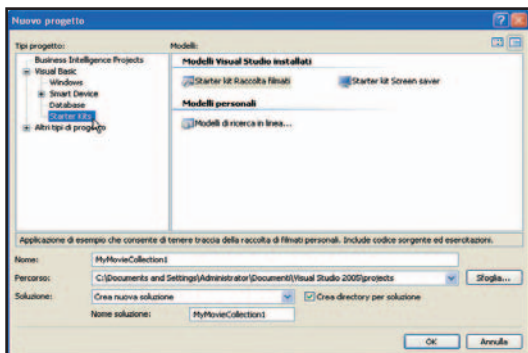
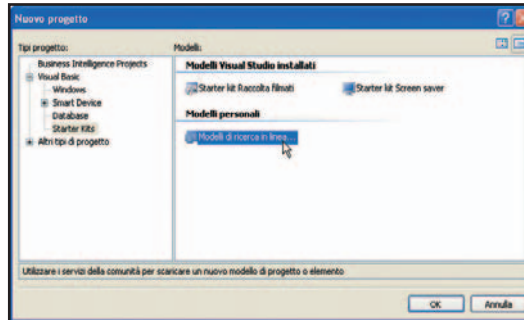
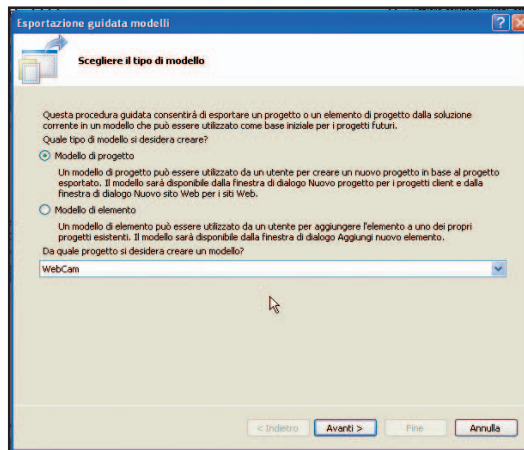
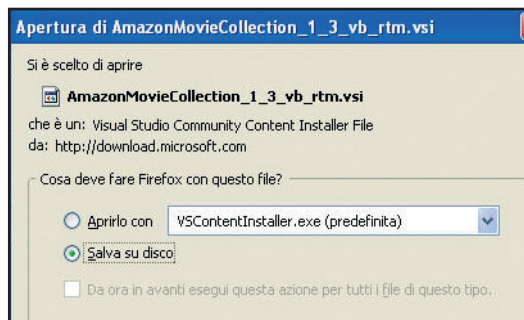
Lo starter kit Screen saver permette di realizzare uno screen saver, visualizzando una sequenza di immagini predefinite o provenienti da una directory indicata dall'utente. Insieme alle immagini viene visualizzato il testo di un RSS (Really Simple Syndication) feed pubblicato in Internet.
Dopo aver generato un nuovo progetto dallo starter kit Screen saver, possiamo effettuare alcune semplici personalizzazioni, seguendo i suggerimenti di programmazione messi a disposizione nella sezione: Estensione dell'applicazione screen saver. In particolare possiamo:

- Selezionare l'immagine predefinita incorporata.
- Impostare ulteriori immagini di sfondo.
- Modificare l'elenco di RSS feed.
- Creare uno screen saver personalizzato.
- Utilizzare servizi Web XML per la visualizzazione di informazioni, ad esempio previsioni del tempo oppure visualizzare le ultime news.

INSTALLARE STARTER KIT DAL WEB

Abbiamo visto come in VB siano disponibili solo due starter kit, dove ne possiamo trovare degli altri? Naturalmente sul Web. Per trovare nuovi starter kit sul Web, si può utilizzare la pagina Cerca della Guida in linea (seguendo la procedura descritta di seguito) oppure possiamo visitare il seguente indirizzo: http://msdn.microsoft.com/vbasic/downloads/starterkits/default.aspx

Il wizard per la ricerca si comporta come segue

1 Selezioniamo la voce di menu File/Nuovo Progetto, in questo modo verrà visualizzata la finestra di dialogo: Nuovo progetto

2 Nella finestra di dialogo Nuovo progetto, dobbiamo selezionare la voce Starter Kits nella parte di sinistra della finestra. Nella parte destra sotto la voce: Modelli personali dobbiamo selezionare Modelli di ricerca in linea e quindi cliccare su OK. In questo modo verrà visualizzata la pagina Cerca della Guida in linea.

3 Nella pagina Cerca della Guida in linea avremo l'opzione Starter kit già selezionata per il filtro Tipo di contenuto. Nella casella di testo possiamo inserire una parola chiave o un nome per lo starter kit da cercare e poi possiamo cliccare su Cerca. I risultati della ricerca verranno raggruppati in base all'origine (Guida locale, MSDN, ecc) e cliccando su un raggruppamento verranno mostrati nell'area Ricerca.

4 A questo punto possiamo cliccare su un risultato per visualizzare ulteriori informazioni sullo starter kit. Dopo aver determinato lo starter kit che si intende utilizzare, dobbiamo leggere ed accettare i termini della licenza Microsoft (End User License Agreement) ed infine possiamo. eseguirne il download.

USARE LA GUIDA

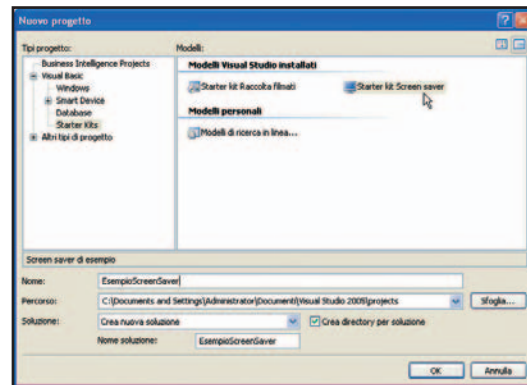
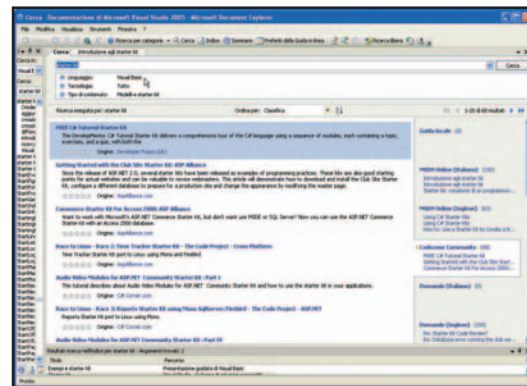
Nella guida in linea di Visual Basic .Net 2005, in corrispondenza di tutti i blocchi di codice, sono presenti alcuni collegamenti contrassegnati con la dicitura Copia codice. Cliccando sul collegamento Copia codice, tutto il codice compreso nel blocco verrà copiato negli Appunti. In questo modo sarà possibile incollare il codice direttamente nell'editor di codice anziché doverlo digitare.

COSA È UN MODELLO?

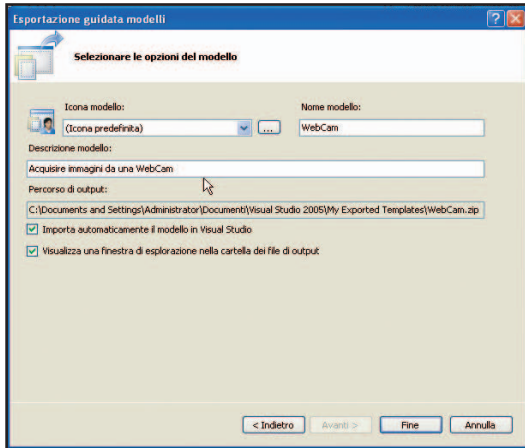
In Visual Studio un modello viene utilizzato come punto di partenza per la creazione di un progetto e contiene le impostazioni, i riferimenti ed i file per iniziare un determinato tipo di progetto.

Dopo aver scaricato lo starter kit desiderato, il passo successivo è la sua installazione. Solo dopo aver installato lo starter kit esso verrà visualizzato nella finestra di dialogo Nuovo progetto.
Di solito uno starter kit ha estensione .vsi così come un qualsiasi altro componente di visual studio, in questo modo VB riconosce automaticamente lo starter kit e lo installa mediante: Visual Studio Content Installer.
Se facciamo doppio click sul file contenente lo starter kit appena scaricato, viene avviato il wizard: Installazione guidata contenuti di Visual Studio, in cui possiamo selezionare gli elementi da installare. Cliccando su Avanti verrà visualizzata la pagina Installazione contenuto e, cliccando su Fine, potremo utilizzare lo starter kit per creare un nuovo progetto VB, seguendo la procedura descritta in precedenza. L'unica differenza è che gli starter kit scaricati saranno visualizzati in Modelli personali. Prima di utilizzare uno starter kit scaricato, da un sito che non sia quello Microsoft, è buona norma determinare l'attendibilità del codice all'esterno di Visual Studio.

CREARE UNO STARTER KIT

Abbiamo descritto come uno Starter kit, per definizione, debba contenere il codice per un'applicazione completa nonché la documentazione atta a spiegare come sia possibile modificare o espandere l'applicazione stessa. Dopo avere realizzato le due componenti fondamentali (codice e documentazione) la creazione di uno starter kit segue la stessa logica della creazione di un modello di progetto normale. L'unica differenza consiste nel fatto che in uno starter kit i file di documentazione, sono impostati in modo da venire aperti quando si crea un progetto basato su di essi.
Come esempio, possiamo utilizzare l'applicazione realizzata e descritta nel numero di Aprile di ioProgrammo, che permette di integrare le immagini di una webcam in una applicazione VB.NET 2005. Per creare lo starter kit non è necessario descrivere nuovamente il funzionamento del codice in dettaglio, ma possiamo utilizzare il codice completo, allegato a questo numero.
Il primo passo per la creazione di uno starter kit, è quello di creare un progetto che venga compilato senza generare errori, quindi nel nostro caso abbiamo già il progetto di partenza
Il secondo passo è quello di produrre la documentazione. Per farlo possiamo riscrivere l'articolo di Aprile in un qualsiasi editor Html, in modo da avere una pagina abbastanza completa con la descrizione dettagliata del codice (anche questa pagina è allegata).
Dopo avere completato il progetto e la documentazione, possiamo creare il modello di progetto per lo starter kit utilizzando: Esportazione guidata modelli e i passi sono i seguenti

1 Selezioniamo la voce di menu File/Esporta Modello, in questo modo si avvia il wizard: Esportazione guidata modelli

2 Nella prima finestra di dialogo, dobbiamo scegliere il tipo di modello. Sono disponibili le voci: Modello di progetto e Modello di elemento. Selezionando la prima voce il modello potrà essere utilizzato come base per creare un nuovo progetto. Selezionando la seconda voce, il modello diventerà un elemento che potrà essere aggiunto ad uno dei propri progetti esistenti, in questo caso il modello sarà visualizzato nella finestra di dialogo: Aggiungi nuovo elemento. Selezioniamo la prima voce e clicchiamo su Avanti

3 Nell'ultima finestra possiamo selezionare un'icona per il modello, tale icona sarà visualizzata nella finestra di dialogo Nuovo progetto. Possiamo, inoltre, assegnare un nome ed una descrizione allo Starter Kit. Dopo

le opportune selezioni possiamo cliccare su Fine. Il progetto verrà compresso ed esportato in un file con estensione .zip ed inserito nella directory specificata. Infine, se abbiamo selezionato l'opzione: Importa automaticamente il modello in Visual Studio, ci ritroveremo in VB il modello appena creato

Se apriamo il file zip generato dal wizard, possiamo notare come sia stato creato un file con estensione .vstemplate

IL FILE DI METADATI .VSTEMPLATE

Ogni starter kit, ed ogni modello, includono un file con estensione vstemplate. Il file con estensione vstemplate racchiude tutti i metadati (in formato XML) che forniscono a VB le informazioni indispensabili per visualizzare il modello nella finestra di dialogo Nuovo progetto e le informazioni necessarie per creare un nuovo progetto dal modello.
I tre componenti fondamentali di un file .vstemplate sono:

- **VSTemplate**. Permette di identificare il modello come modello di progetto o di elemento in base al valore dell'attributo Type, ed indica il numero di versione del modello per mezzo dell'attributo Version. In Visual Studio 2005 i modelli hanno un valore dell'attributo Version pari a 2.0.0.
- **TemplateData**. Permette di indicare i dati che classificano il modello e di definire come deve essere visualizzato il modello, nella finestra di dialogo Nuovo progetto o Aggiungi nuovo elemento.
- **TemplateContent**. Permette di indicare tutti i file inclusi nel modello.

La struttura può, quindi, essere riassunta in questo modo:

```
<VSTemplate Type="TemplateType" Version="x.x.x">
  <TemplateData>    </TemplateData>
  <TemplateContent>    </TemplateContent>
  ...
</VSTemplate>
```

Di particolare importanza, nella sezione TemplateContent riveste l'attributo Project che indica, in dettaglio, tutti i file o le directory da aggiungere al progetto. All'interno di Project, troviamo il parametro figlio ProjectItem.

ProjectItem ammette i seguenti attributi facoltativi:

- **TargetFileName**. Permette di indicare il nome e il percorso di ogni elemento quando viene creato un progetto dal modello. È possibile utilizzare l'attributo TargetFileName per creare una struttura di directory diversa dalla struttura di directory presente nel file .zip del modello.
- **ReplaceParameters**. È un valore booleano che permette di indicare se nell'elemento specificato, i valori dei parametri dovranno essere sostituiti nel momento in cui viene creato un progetto dal modello. Il valore predefinito è false.
- **OpenInEditor**. È un valore booleano che permette di indicare se l'elemento specificato, dovrà essere aperto nell'editor di VB quando viene creato un progetto dal modello. Il valore predefinito è false.
- **OpenInWebBrowser**. È un valore booleano che permette di indicare se l'elemento specificato, dovrà essere aperto nel browser Web quando viene creato un progetto dal modello. Nel browser è possibile aprire solo i file HTML ed i file di testo locali del progetto. Non è possibile aprire URL esterni. Il valore predefinito è false.
- **OpenInHelpBrowser**. È un valore booleano che permette di indicare se l'elemento specificato, dovrà essere aperto nel visualizzatore della Guida quando viene creato un progetto dal modello. Nel browser della Guida è possibile aprire solo i file HTML ed i file di testo locali del progetto. Non è possibile aprire URL esterni. Il valore predefinito è false.
- **OpenOrder**. Permette di indicare un valore

DOVE TROVO LA DOCUMENTAZIONE?

Un modello, di solito, non contiene la documentazione. Generalmente il codice contiene alcuni commenti come supporto allo sviluppo, ma la documentazione con le istruzioni per sviluppare un'applicazione tramite il modello è molto limitata. Uno starter kit, invece, contiene la documentazione dettagliata che spiega i vari componenti del progetto, oppure una guida dettagliata per eventuali personalizzazioni a completamento del progetto stesso.

numerico che rappresenta l'ordine in cui gli elementi verranno aperti nei rispettivi editor. Tutti i valori devono essere multipli di 10. Ad essere aperti per primi saranno gli elementi con valori OpenOrder inferiori.

AVVIARE LA DOCUMENTAZIONE ALL'APERTURA

Per completare la creazione del nostro Starter kit, dobbiamo fare in modo che la prima pagina visualizzata, all'apertura, sia quella relativa alla documentazione.
Per ottenere questo effetto, dobbiamo aprire il file con estensione .vstemplate con editor qualsiasi, ed individuare l'elemento **ProjectItem** che contiene il file di documentazione dello Starter kit.

Abbiamo descritto come sia possibile aprire un file nei seguenti modi:

- nell'editor di codice ponendo *OpenInEditor="true"*
- nel browser Web ponendo *OpenInWebBrowser="true"*
- nel visualizzatore della Guida ponendo *OpenInHelpBrowser="true"*

Nel nostro caso dobbiamo, quindi, impostare l'attributo **OpenInWebBrowser** su true.
È possibile, inoltre, specificare più file e anche l'ordine in cui dovranno essere aperti, nel nostro caso poniamo l'attributo Open Order="10" per fare in modo che la documentazione sia visualizzata per prima pertanto l'istruzione completa che dovremo scrivere sarà:

```
<ProjectItem OpenOrder="10"
        OpenInWebBrowser="true">Doc\Webcam.htm
                                    </ProjectItem>
```

Ed il file MyTemplate.vstemplate diventerà:

```
<VSTemplate Version="2.0.0"
 xmlns="http://schemas.microsoft.com/developer/vst
                    emplate/2005" Type="Project">
  <TemplateData>
    <Name>WebCam</Name>
    <Description>Acquisire immagini da una
                              WebCam</Description>
    <ProjectType>VisualBasic</ProjectType>
    <ProjectSubType>
    </ProjectSubType>
    <SortOrder>1000</SortOrder>
    <CreateNewFolder>true</CreateNewFolder>
    <DefaultName>WebCam</DefaultName>
<ProvideDefaultName>true</ProvideDefaultName>
    <LocationField>Enabled</LocationField>
<EnableLocationBrowseButton>true</EnableLocation
                                   BrowseButton>
    <Icon>__TemplateIcon.ico</Icon>
  </TemplateData>
  <TemplateContent>
    <Project TargetFileName="WebCam.vbproj"
    File="WebCam.vbproj" ReplaceParameters="true">
      <ProjectItem ReplaceParameters="true"
               TargetFileName="AssemblyInfo.vb">
                  AssemblyInfo.vb</ProjectItem>
      <ProjectItem ReplaceParameters="true"
                TargetFileName="Definizioni.vb">
                  Definizioni.vb</ProjectItem>
      <Folder Name="doc" TargetFolderName="doc">
        <ProjectItem OpenOrder="10"
           OpenInWebBrowser="true">WebCam.htm
                                  </ProjectItem>
      </Folder>
      <ProjectItem ReplaceParameters="true"
            TargetFileName="Form1.vb">Form1.vb
                                  </ProjectItem>
      <ProjectItem ReplaceParameters="true"
          TargetFileName="Form1.resx">Form1.resx
                                  </ProjectItem>
      <Folder Name="My Project"
                  TargetFolderName="My Project">
        <ProjectItem ReplaceParameters="true"
   TargetFileName="Resources.resx">Resources.resx
                                  </ProjectItem>
        <ProjectItem ReplaceParameters="true"
          TargetFileName="Resources.Designer.vb">
             Resources.Designer.vb</ProjectItem>
      </Folder>
    </Project>
  </TemplateContent>
</VSTemplate>
```

Luigi Buono

NOTA

CATEGORIE DI MODELLI

È possibile organizzare i modelli installati in base alle proprie esigenze creando delle sottocategorie personalizzate all'interno della cartella del linguaggio di programmazione. Per creare le sottocategorie è sufficiente creare le sottodirectory corrispondenti, in questo modo nelle finestre di dialogo Nuovo progetto e Aggiungi nuovo elemento le sottocategorie verranno rappresentate come cartelle virtuali all'interno di ciascun linguaggio.

INSTALLARE LO STARTER KIT

Affinché i modelli possano essere mostrati nelle finestre di dialogo Nuovo progetto e Aggiungi nuovo elemento, i file che lo compongono devono essere inseriti in uno dei due percorsi predefiniti. Per impostazione predefinita, i modelli installati con VB risiedono nei seguenti percorsi:

<VisualStudioInstallDir>\Common7\IDE\ItemTemplates\VisualBasic\Locale
<VisualStudioInstallDir>\Common7\IDE\ProjectTemplates\VisualBasic\Locale

E gli Starter Kit in:

<VisualStudioInstallDir>\Common7\IDE\ProjectTemplates\VisualBasic\Starter Kits

Se in questi percorsi esiste un file compresso che contiene un file con estensione vstemplate, nelle finestre di dialogo verrà visualizzato il modello corrispondente.

SERVIZI DI SISTEMA FACCIAMOLI IN JAVA

PANORAMICA DELLA NUOVA VERSIONE DI JAVA SERVICE WRAPPER, STABILE E FACILE DA PROGRAMMARE. ECCO LA LIBRERIA OPEN SOURCE PER GESTIRE APPLICAZIONI J2SE COME SERVIZI NT O DEMONI UNIX

Java Service Wrapper è un software Open Source, distribuito sotto licenza MIT, scritto in Java e C, che nasce come utility per la gestione di applicazioni Java sotto forma di Windows NT Services o Unix daemons. Il progetto, che vide le prime luci nell'anno 1999 per mano di Leif Mortenson, un giovane ingegnere della Silver Egg Technology, a quel tempo proprietaria del software, era stato inizialmente concepito per gestire applicazioni Java unicamente come NT Services. Il suo funzionamento era caratterizzato dal caricamento della libreria *"jvm.dll"* e dall'esecuzione di una nuova istanza di Java Virtual Machine eseguita via JNI (Java Native Interface). Come spesso succede con i tool di questa tipologia, le prime versioni furono caratterizzate da sintomi di instabilità a causa di crash della JVM causati da errori di tipo "HotSpot Access Violation". I progressi più significativi si sono avuti con il passaggio della libreria da applicazione proprietaria a Open Source e la successiva registrazione del progetto su SourceForge (2001), il portale degli "amanti" del software libero. Da allora si sono susseguiti continui miglioramenti e sono state implementate nuove features che hanno fatto diventare Java Service Wrapper una "preziosa" risorsa soprattutto per chi ha a che fare con applicazioni server e più in generale con i cosiddetti software 7x24 (costantemente in esecuzione, 7 giorni la settimana per 24 ore al giorno). Il tool, come vedremo nel prosieguo dell'articolo, può essere utilizzato come *monitor* della JVM e può prendere delle decisioni a fronte di situazioni di errore della stessa. Prenderemo in esame l'ultima versione stabile rilasciata, la 3.2.1, scaricabile dal sito http://wrapper.tanukisoftware.org, disponile per le seguenti piattaforme: AIX, FreeBSD, HP-UX, SGI Irix, Linux, Macintosh OS X, DEC OSF1, Sun Solaris ed ovviamente Microsoft Windows. Il progetto è distribuito sotto forma di file compresso al cui interno possiamo trovare gli elementi necessari per integrazione all'interno delle nostre applicazioni.

PROCESSO DI INTEGRAZIONE

L'integrazione di Java Service Wrapper all'interno di un'applicazione può avvenire attraverso l'utilizzo di tre differenti metodologie. La scelta varia in base al funzionamento dell'applicazione stessa. Il primo metodo è indicato per i software che terminano il loro ciclo di vita attraverso il comando CTRL+C o l'invocazione del metodo *System.exit()*; il secondo è destinato ad applicazioni avviate e stoppate tramite l'invocazione di due differenti classi, come ad esempio Tomcat (http://jakarta.apache. org/tomcat/); mentre il terzo è consigliato per rendere il software più affidabile e capace di rispondere ad eventuali malfunzionamenti della JVM. In tutti i casi l'integrazione è legata a dei particolari file di properties indipendenti dalla piattaforma mentre la creazione e rimozione del relativo servizio sono agevolate da un set di script standard compresi nella libreria. Per comprendere al meglio il funzionamento del tool, illustreremo come integrare Java Service Wrapper all'interno di uno dei più diffusi Application Server Open Source: JBoss (http://www.jboss.org/). Utilizzeremo il primo metodo che, oltre ad essere quello usato più di frequente, si sposa benissimo alle esigenze di un'applicazione server come JBoss. È importante sottolineare che questa tipologia di integrazione non è invasiva in quanto non richiede alcuna modifica al codice esistente.

JAVA SERVICE WRAPPER CONFIGURATION

Per illustrare il processo di integrazione di Java Service Wrapper useremo la versione 3.2.7 di JBoss. Gli step che di seguito elencheremo sono abbastanza generici e, con le modifiche dovute, potranno essere facilmente applicati sia a differenti versioni dell'application server che ad altri software. Come già accennato precedentemente, il funzionamento di Java Service Wrapper si basa

su specifici files di properties. Per poterli utilizzare su piattaforme diverse supporremo che le variabili d'ambiente *JAVA_HOME, JBOSS_HOME e JSW_HOME*, che rappresentano rispettivamente le directory d'installazione della JVM, dell'application server e del wrapper, siano presenti nella configurazione del nostro sistema.
Per agevolare il processo di integrazione, la libreria ci viene incontro con una serie di files base (identificati dal suffisso .in) tra cui *wrapper.conf.in*, contenuto nella cartella JSW_HOME/*src/conf*. Quest'ultimo contiene le proprietà standard da cui partire per creare una configurazione funzionante. Il primo passo consiste nel copiare questo file nella directory *JBOSS_HOME/conf* (da creare nel caso non sia presente), eliminando l'estensione *.in*. A questo punto non ci rimane che editare il file *wrapper.conf* e modificare le seguenti proprietà:

```
wrapper.java.command=%JAVA_HOME%/bin/java
wrapper.java.mainclass=org.tanukisoftware.wrapp
er.WrapperSimpleApp
wrapper.java.classpath.1=%JBOSS_HOME%/lib/wr
apper.jar
wrapper.java.classpath.2=%JBOSS_HOME%/bin/run.jar
wrapper.java.classpath.3=%JAVA_HOME%/lib/tools.jar
wrapper.java.library.path.1=%JBOSS_HOME%/lib
wrapper.app.parameter.1=org.jboss.Main
```

- **wrapper.java.command**: rappresenta il path completo dell'eseguibile necessario ad eseguire la JVM. Nel caso in cui si voglia utilizzare il comando java presente nel PATH di sistema basta specificare il valore java.
- **wrapper.java.mainclass**: è la classe di partenza eseguita dal wrapper. Deve obbligatoriamente implementare l'interfaccia *WrapperListener* e deve garantire l'inizializzazione della classe *WrapperManager* (entrambe contenute nel package *org.tanukisoftware.wrapper*). Per il nostro esempio useremo una classe helper fornita con il tool che funge da proxy per l'effettivo avvio dell'applicazione. Questo tipo di approccio è molto consigliato in quanto non è invasivo.
- **wrapper.java.classpath.<n>**: indica gli elementi da inserire nel classpath dell'applicazione. È obbligatorio includere il file *wrapper.jar*. Questo tipo di proprietà è incrementale, pertanto per l'inserimento di un elemento aggiuntivo è necessario incrementare l'indice presente nella parte finale della chiave. Per il nostro esempio aggiungeremo altre due librerie necessarie per il corretto funzionamento dell'application server.
- **wrapper.java.library.path.<n>**: rappresenta il library path della JVM. Deve includere il path nel quale posizioneremo le librerie native fornite insieme al tool.
- **wrapper.app.parameter.<n>**: rappresenta un parametro da passare al wrapper quando questo viene lanciato. Quando il valore della proprietà wrapper.java.mainclass è impostato a *org.tanukisoftware.wrapper.WrapperSimpleApp* il primo parametro applicativo deve essere la classe di avvio dell'applicazione.

Per includere le classi della libreria nel classpath è necessario copiare il file *wrapper.jar*, contenuto nella cartella *JSW_HOME/lib*, nella directory *JBOSS_HOME/lib*. Infine, per garantire la creazione del log file, utile a tracciare le operazioni del wrapper, dovremo creare una directory *logs* all'interno della *JBOSS_HOME*.
Fino ad adesso ci siamo soffermati sulla configurazione generale di Java Service Wrapper; nelle prossime sezioni descriveremo invece le attività specifiche per la creazione di NT Services e Unix daemons. Per una visione completa delle proprietà messe a disposizione dal tool è consigliabile consultare la documentazione, disponibile in formato HTML, contenuta all'interno della cartella *JSW_HOME/docs*.

NT SERVICES PROPERTIES

In ambiente Windows, per completare la fase di configurazione, sarà sufficiente modificare le seguenti proprietà:

```
wrapper.ntservice.name=JBoss
wrapper.ntservice.displayname=JBoss Application
Server 3.2.7
wrapper.ntservice.description=Servizio di avvio di
JBoss 3.2.7 (powered by Java Service Wrapper)
wrapper.ntservice.starttype=AUTO_START
wrapper.ntservice.interactive=false
```

In questo modo il wrapper creerà un servizio NT con nome JBoss, eseguito in background ed avviato alla partenza del sistema. Infine, i seguenti passi consentiranno di perfezionare il processo di integrazione:

- copia della libreria nativa *JSW_HOME/Wrapper.DLL* nella directory JBOSS_HOME/lib.
- copia dell'eseguibile *JSW_HOME/bin/Wrapper.exe* in *JBOSS_HOME/bin*.
- copia dei file App.bat.in, *InstallApp-NT.bat.in*, *UninstallApp-NT.bat.in*, contenuti nella cartella *JSW_HOME/src/bin*, in *JBOSS_HOME/bin*.
- eliminazione dell'estensione .in per i file copiati nel punto precedente.

Eseguendo lo script *JBOSS_HOME/bin/InstallApp-NT.bat* JBoss verrà aggiunto alla lista dei servizi NT (vedi Fig.1).

UNIX DAEMONS PROPERTIES

La creazione di uno Unix daemon necessita della presenza di un file aggiuntivo, sh.script.in, contenuto nella cartella *JSW_HOME/src/bin*. Questo file va copiato nella cartella *JBOSS_HOME/bin* e rinominato in jboss. Inoltre, è consigliabile impostare al suo interno le seguenti variabili:

```
APP_NAME="JBoss"
APP_LONG_NAME="JBoss Application Server 3.2.7"
```

Analogamente al processo di creazione dei servizi NT, dovremmo eseguire i seguenti step:

- copia della libreria nativa *JSW_HOME/libwrapper.so* nella directory *JBOSS_HOME/lib*.
- copia del file eseguibile *JSW_HOME/bin/wrapper* in *JBOSS_HOME/bin*.

A questo punto non ci rimane che aggiungere un nuovo servizio costituito da un link simbolico che punta allo script *jboss* precedentemente creato. I dettagli di quest'ultima attività non sono stati descritti in quanto molte distribuzioni di sistemi operativi Unix like presentano delle differenze a riguardo.

JVM SOTTO CONTROLLO

L'ingegneria del software definisce un'applicazione affidabile quando questa opera in modo corretto durante il suo tempo di esecuzione. Le applicazioni server, vista la loro complessità, sono solitamente più esposte a comportamenti anomali. Nel caso degli application server il problema è ancora più delicato in quanto il loro funzionamento è influenzato da applicazioni di terze parti. L'introduzione di Java Service Wrapper può anche fungere da monitor della JVM per controllarne la corretta esecuzione. La JVM è eseguita come processo separato ed è interrogata periodicamente via socket, con polling time configurabile dalla proprietà *wrapper.ping.interval*. Ogni evento anomalo generato dalla JVM è processato da un listener, rappresentato da una classe che implementa l'interfaccia *WrapperListener*, che ne esegue l'azione corrispondente (stop, restart, messaggio di log, ecc.). Per comprendere velocemente questa utile caratteristica è consigliabile eseguire lo script *JSW_HOME/bin/TestWrapper.bat* (.sh per piattaforme Unix like). Attraverso questa applicazione di esempio è possibile simulare la generazione di un evento inaspettato della JVM (halt, crash, access violation, ecc.) e visualizzare il comportamento del wrapper.

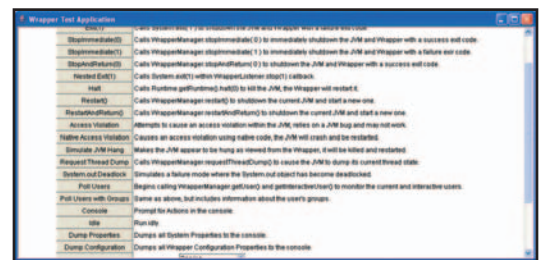



Fig. 2: *GUI dell'applicazione di esempio per la simulazione degli eventi*

Un'altra interessante funzionalità, utile per accrescere l'affidabilità delle nostre applicazioni, consiste nella possibilità di impostare dei filtri sull'output della JVM. Ad esempio, aggiungendo al file *wrapper.conf* le seguenti proprietà:

```
wrapper.filter.trigger.1=java.lang.OutOfMemoryError
wrapper.filter.action.1=RESTART
```

istruiremo Java Service Wrapper a riavviare l'istanza della JVM monitorata nel caso in cui si verifichi la saturazione della heap memory. È possibile aggiungere più coppie di trigger/action incrementando l'indice posto alla fine del nome della proprietà. Le azioni configurabili sono: *RESTART, SHUTDOWN* e *NONE*.

CONCLUSIONI

In questo articolo abbiamo visto come, attraverso l'integrazione di Java Service Wrapper, sia possibile gestire le nostre applicazioni come NT Services o Unix daemons. Questa soluzione è anche utile per gestire comportamenti anomali della JVM soprattutto quando si gestiscono applicazioni critiche come quelle server.

Fabrizio Fortino

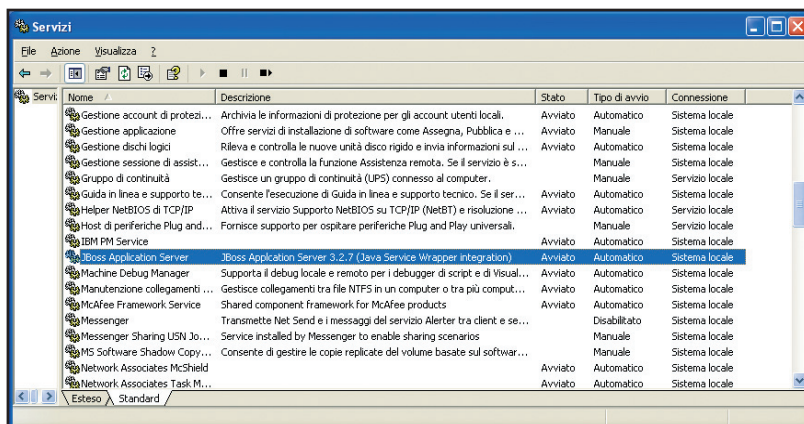



Fig. 1: *JBoss Application Server nella lista degli NT services di Windows*

GERARCHIE CON MS SQL SERVER 2000

SPESSO NELLE NOSTRE ATTIVITÀ QUOTIDIANE CI CAPITA DI DOVER LAVORARE CON DATI ORGANIZZATI IN MODO GERARCHICO. SCOPRIAMO GLI STRUMENTI CHE CI VENGONO MESSI A DISPOSIZIONE DA .NET E MS SQL SERVER 2000 PER GESTIRE AL MEGLIO QUESTE SITUAZIONI

CD WEB
gerarchie_con_MS_SQL_SERVER_2000.zip
cdrom.ioprogrammo.it

REQUISITI

Conoscenze richieste
Conoscenze medie di .Net Framework (C#), conoscenze avanzate di MS Sql Server 2000

Software
Net Framework 1.1 e sup., MS Sql Server 2000 e sup.
Impegno: 9 (da 1 a 10)

Impegno

Tempo di realizzazione

Prima di entrare nel vivo dell'articolo, oggi vi voglio proporre un piccolo gioco. Supponete che vi troviate davanti a SQL Server ed abbiate sotto mano una tabella così composta

Id_persona	Nome	Id_padre
1	Giovanni	<NULL>
2	Enrico	1
3	Alessandro	2
4	Francesco	1

Diciamo che l'analista che l'ha sviluppata non era un genio, e che ora vi tocca estrapolare da questa tabella tutti i record relativi ai nipoti di "Giovanni".
L'apparente banalità del quesito, tuttavìa, svela ben presto la sua vera natura "subdola" e intrigante: se i campi Id_persona e Id_padre fossero appartenenti a tabelle diverse, non avremmo avuto esitazioni ad usare un **inner join** filtrato da una **where-condition**, ma in questo caso questa soluzione sembra essere decisamente improponibile. Una prima soluzione potrebbe essere

```
select * from tb_persone where id_padre in
(select id_persona from tb_persone where id_padre =
(select id_persona from tb_persone where id_persona
= 1)
)
```

Ora, questa soluzione è corretta, ma non è molto efficiente. Sarebbe invece opportuno usare i **join**. Ma come fare quando c'è un'unica tabella?

SELF-JOIN

Ecco la soluzione dell'enigma. In questo caso facciamo riferimento ad una tabella unica, in cui figura una chiave primaria, che identifica univocamente ogni singolo record, un campo di descrizione e una chiave secondaria. Quest'ultima, paradossalmente, fa riferimento non ad un campo appartenente ad un'altra tabella, ma allo stesso campo chiave primaria appena citato. Visto che ad ogni padre possono corrispondere più figli, ma ad ogni figlio corrisponde sempre un solo padre (fino a prova contraria !), possiamo quindi creare un vero e proprio **self-join con relazione riflessiva** che simuli una normale relazione **one-to-many** (uno a molti) tra due tabelle. Per far ciò, è sufficiente aggiungere la tabella all'interno di un diagramma, cliccare sul campo chiave primaria e, tenendo premuto il pulsante sinistro del mouse, spostare il puntatore sul campo che vogliamo far diventare chiave esterna (d'altronde, non seguiremmo la stessa procedura se volessimo mettere in relazione due tabelle distinte ?). Come per magia appariranno sul legame creato i due simboli classici della relazione uno-a-molti (la chiave e l'infinito), e si attiveranno i vincoli di integrità referenziale predefiniti.

Fig. 1: La relazione riflessiva rappresentata in SQL Server

Vogliamo, a questo punto, creare una vista che estragga tutti i record relativi ai nipoti di Giovanni (il cui id_persona è 1):

```
select nipoti.id_persona, nipoti.nome, figli.id_persona
as id_persona_padre, figli.nome as nome_padre, pa
dri.id_persona as id_persona_nonno, padri.nome as
nome_nonno
from tb_persone nipoti inner join
(tb_persone padri inner join tb_persone figli on
```

```
padri.id_persona=figli.id_padre)
            on nipoti.id_padre=figli.id_persona
 where padri.id_persona=1
```

Come si può facilmente osservare, gli alias **padri**, **figli** e **nipoti** servono a richiamare tre istanze diverse della stessa tabella, al fine di consentire la corretta impostazione degli *inner join*. Gli alias sono necessari anche per ridefinire i campi del risultato della vista che hanno lo stesso nome (**figli.id_persona, figli.nome, padri.id_persona** e **padri.nome**), anche se appartengono ad istanze diverse. Guardando la vista salvata nel database in modalità Design View (vedi Figura 2), ci accorgiamo subito che SQL Server, a fronte del codice SQL da noi inserito, crea graficamente tre tabelle distinte nominate con gli alias da noi assegnati alle istanze della tabella Tb_persone.

UN CASO PIÙ COMPLESSO

Consideriamo una piccola impresa manifatturiera produttrice di semilavorati destinati all'industria. Ammettendo che il semilavorato finale A, ad esempio, sìa composto a sua volta dai semilavorati B e C, in questo caso non possiamo tuttavia affermare che B e C appartengono esclusivamente ad A: al contrario, è probabile che essi "concorrano" anche alla distinta base di D, e così via discorrendo. Insomma, è evidente che in questo caso ci troviamo di fronte ad un tipo particolare di relazione many-to-many (molti a molti), in cui entrambe le chiavi esterne fanno riferimento ad un unico campo chiave primaria relativo ad una sola tabella. Le tabelle in questione potrebbero essere le seguenti quelle rappresentate in Figura 2 e 3.
Il diagramma che rappresenta questa situazione è il seguente è rappresentato in figura 4.
A questo punto, finalmente, possiamo cominciare ad introdurre le tecniche di programmazione utili per sfruttare le particolari relazioni appena illustrate.

FACCIAMOLO CON .NET

Ora che abbiamo dato un'occhiata all'argomento da un punto di vista generale, non ci resta che analizzare gli strumenti messi a disposizione da .Net Framework e Sql Server al fine di consentire un corretto approccio con esso. Il nostro scopo è, principalmente, quello di realizzare degli algoritmi che consentano di modellare graficamente, attraverso una comoda struttura ad albero, le gerarchie relative agli esempi

Fig. 2: *La vista salvata in modalità design*

Column Name	Data Type	Length	Allow Nulls
id_semilavorato	int	4	
decrizione	varchar	50	✓

Fig. 3: *La tabella che rappresenta un semilavorato*

Column Name	Data Type	Length	Allow Nulls
id_semilavorato	int	4	
id_componente	int	4	
quantita	int	4	✓

Fig. 4: *La tabella di un semilavorato con relazione sulla tabella proimaria*

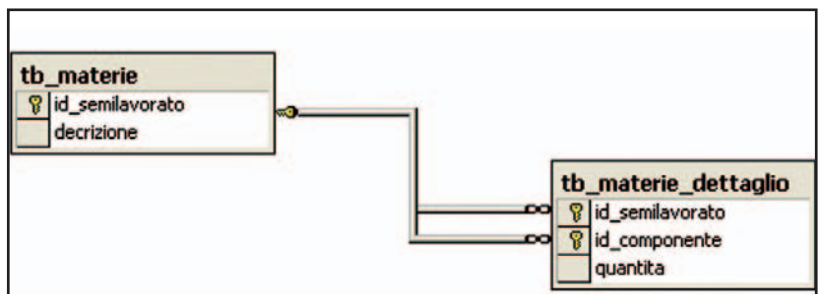



Fig. 5: *Il diagramma che rappresenta la relazione fra le tabelle*

considerati.
Vista la complessità degli obiettivi che ci siamo preposti, la via più comoda da seguire sarà sicuramente quella ricorsiva. Una funzione si dice ricorsiva quando richiama se stessa; la logica che adotterò, quindi, è la seguente: creo un controllo Treeview su un form, quindi genero il primo ramo e chiamo una funzione che, tramite l'invocazione di un comando (o di una stored procedure), estrae i sottorami relativi a quello preso in considerazione, aggiungendoli al controllo. Per ciascuno di questi sottorami, inoltre, deve essere richiamata sempre la stessa funzione, allo scopo di esaurire tutti i vari livelli di cui si compone la gerarchia.

```
using System;
```

```
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using Progetto_fabbrica.connessioni_db;
namespace Progetto_fabbrica
{
    public class Albero :
            System.Windows.Forms.Form
    {
        private
    System.Windows.Forms.TreeView treeView1;
        private System.ComponentMo
          del.Container components = null;
        private fornisci_connessione f;
        public Albero()
        {
            InitializeComponent();

        }
        protected override void Dispose(
                bool disposing )
        {
            if( disposing )
            {
                if(components
                    != null)
                {
    components.Dispose();
                }
            }
            base.Dispose( disposing
                );
        }
        [STAThread]
        static void Main()
        {
            Application.Run(new Al
                bero());
        }
        #region Windows Form Designer
                generated code
        private void InitializeComponent()
        {
            this.treeView1 = new
        System.Windows.Forms.TreeView();
            this.SuspendLayout();
            //
            // treeView1
            //
            this.treeView1.ImageIn
                dex = -1;
            this.treeView1.Location
        = new System.Drawing.Point(40, 32);
            this.treeView1.Name =
                "treeView1";
            this.treeView1.Selecte
                dImageIndex = -1;
            this.treeView1.Size =
        new System.Drawing.Size(928, 528);
            this.treeView1.TabIndex
                = 0;
            //
            // Albero
            //
            this.AutoScaleBaseSize
        = new System.Drawing.Size(5, 13);
            this.ClientSize = new
        System.Drawing.Size(992, 590);
this.Controls.Add(this.treeView1);
            this.Name = "Albero";
            this.Text = "Albero";
            this.Load += new Sy
        stem.EventHandler(this.Albero_Load);
            this.ResumeLayout(fal
                se);
        }
        #endregion
        private void Albero_Load(object
            sender, System.EventArgs e)
        {
            this.f=new fornisci_con
    nessione("server=BUREAU-65284037;Integrated Se
    curity=SSPI;database=semilavorati; User id=enri
        chetto; password=parapippo");
            TreeNode radice=new
                TreeNode("Inizio");
this.treeView1.Nodes.Add(radice);
            Aggiungi_nodi(1, radi
                ce);
            // con questo parametro
                setto il punto
```

INDICI CLUSTERED

Un indice clustered è un indice che esegue il sorting fisico dei record, influenzandone direttamente la disposizione all'interno della tabella: esso è utile quindi per velocizzare le ricerche che devono avvenire all'interno di un determinato range di valori. La chiave primaria è predefinitamente un indice clustered, ma anche un campo che ammette valori duplicati può essere associato ad un indice di questo tipo. Alcune note sulla scelta degli indici nel caso della tabella che accoglie le due chiavi esterne, con riferimento alla relazione molti a molti: la creazione di una chiave primaria multicampo mi consente di rendere i record univoci per i campi interni alla chiave, e obbliga comunque l'utente a inserire sempre qualcosa (la chiave primaria non ammette valori NULL). Se volevo consentire l'immissione di valori NULL allora avrei creato un Costraint Unique multicampo, che mi consentiva, tuttavìa, di inserire soltanto un valore NULL per chiave, visto che già due valori uguali violano il vincolo stesso. Se non si vuole inserire una chiave primaria, conviene sicuramente impostare comunque un indice clustered sul campo che viene interrogato dalle query, anche se tale campo, come nel nostro caso, contiene valore duplicati.

```
                    // dal quale l'applicazio
               ne deve cominciare a sviluppare
                    // la gerarchia
                    treeView1.ExpandAll();
          }
          private void Aggiungi_nodi(int
               valore, TreeNode nodo_parente)
          {
                    DataSet set_dati=
          f.risultato_dataset_con_sp(valore);
                    // la chiamata al metodo
               di cui sopra estrae tutti i record
                    // figli dell'id_persona
          passato con valore e restituisce un
                    // dataset
                    foreach (DataRow riga
                                         in
          set_dati.Tables["tb_persone"].Rows)
                    {
                         string stringa;
                    int id_persona;
          stringa=(string) riga["nome"];
                         TreeNode
  nuovo_nodo=new TreeNode((string) riga["nome"]);
id_persona=(int) riga["id_persona"];
                         // prelevo
     l'id_persona e richiamo nuovamente la funzione
                    nodo_parente.
                    Nodes.Add(nuovo_nodo);
                         Aggiungi_
               nodi(id_persona, nuovo_nodo);
                    }
          }
     }
}
```

Ricordo agli sviluppatori Visual Basic .Net che il parametro **nodo_parente** della funzione Aggiungi_nodi deve essere passato per riferimento (quindi con **ByRef**, invece che con ByVal), visto che deve riferirsi al primo oggetto Node dell'albero che deve essere via via popolato. Brutte notizie per chi è solito usare il veloce cursore Forward-only DataReader per interrogare il database: la connessione, infatti, una volta creato il DataReader, rimane sotto il suo controllo esclusivo fino a che questo non viene chiuso, e se si prova a creare un nuovo cursore essa emetterà la seguente eccezione: "**There is already an open DataReader associated with this Connection which must be closed first**". Per ovviare a tale inconveniente, pertanto, utilizzeremo gli oggetti DataSet, che sono un po' più lenti, visto che memorizzano in cache i risultati delle query, ma non hanno il limite sopra citato. Il DataSet, per i nostri scopi, rappresenta comunque la soluzione ottimale, non soltanto perché si presta con facilità ad essere impiegato in una funzione ricorsiva, ma anche perché consente di creare degli oggetti DataRelation che stabiliscono automaticamente una gerarchia all'interno dei dati del resultset. Il DataRelation, inoltre, non funziona soltanto con i DB relazionali, ma può essere usato anche con i file XML, ovviamente soltanto se questi rispettano la struttura imposta dalla gerarchia rappresentata dal DataRelation. Nell'esempio qui sotto, in particolare, mi avvalgo degli oggetti DataSet abbinati ai DataRelation per creare un albero ricorsivo alimentato da DB:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Data;
namespace progetto_DB
{
          public class Form1 :
                         System.Windows.Forms.Form
          {
                    private
          System.Windows.Forms.TreeView treeView1;
                    private System.ComponentMo
                         del.Container components = null;
                    private DataSet _ds;
                    private const string GERARCHIA =
                                        "parentela";
                    public Form1()
                    {
                              InitializeComponent();
                              string cString = "Persist
   Security Info=False;Integrated Security=SSPI;data
  base=semilavorati;server=BUREAU-65284037; User
                  id=enrico; password=passaparola;";
                              SqlConnection conn =
                         new SqlConnection(cString);
                              SqlCommand selectCMD
       = new SqlCommand("select * from tb_persone",
                                             conn);
                              SqlDataAdapter persDA
```

INDICI NON CLUSTERED

Nella prima tabella, dove esiste una relazione riflessiva uno a molti, mi converrà tuttavia inserire un indice non clustered sul campo Molti per velocizzare il reperimento dei record. Un indice nonclustered non influenza l'ordine di visualizzazione dei record della tabella, e consente di effettuare ricerche veloci, avendo i campi su cui si effettua la ricerca ordinati e legati ad un eventuale indice clustered (se questo non è presente, SQL Server genera una chiave univoca per ogni record e la associa alle chiavi dell'indice).

```
= new SqlDataAdapter();
persDA.SelectCommand   = selectCMD;
_ds = new DataSet("tb_persone");
persDA.Fill(_ds, "tb_persone");
_ds.Relations.Add(GE
RARCHIA, _ds.Tables["tb_persone"].Columns[0],
_ds.Tables["tb_persone"].Columns[2]);
// la colonna 0 rappresenta l'uno,
//la colonna 2 rappresenta il molti
TreeNode radice = new
TreeNode("Nodo principale");
Aggiungi_nodo(_ds.T
bles[0].Rows[0], radice);
this.treeView1.Nodes.Add(radice);
this.treeView1.Expan
dAll();
}
private void Aggiungi_nodo(Data
Row riga_parente, TreeNode nodo_parente) {
foreach (DataRow dr in riga_parente.
GetChildRows(GERARCHIA)) {
if (dr[0].ToString() != "0") {
string stringa=(string) dr[1];
TreeNode nuovo_nodo= new
TreeNode(stringa);
nodo_parente.Nodes.Add(nuovo_nodo);
Aggiungi_nodo(dr, nuovo_nodo);
}
}
}
. . .
```

In questo esempio, invece, sfrutterò un oggetto DataSet abbinato a DataRelation per creare un albero alimentato da file XML:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace progetto_XML
{
public class Form1 :
System.Windows.Forms.Form
{
private
System.Windows.Forms.TreeView treeView1;
private System.ComponentMo
del.Container components = null;
private DataSet _ds;
private const string ALBERO =
"gerarchia";
public Form1()
{
InitializeComponent();
_ds = new DataSet();
_ds.ReadXml
(Application.StartupPath + "/gerarchia.xml");
_ds.Relations.Add(
ALBERO, _ds.Tables[0].Columns[0], _ds.Tables[0].
Columns[1]);
//la colonna 0 rappresenta
//la chiave primaria  (parent), la
// colonna 1 la chiave secondaria
// (child)
TreeNode radice = new
TreeNode("Nodo principale");
Aggiungi_nodo(_ds.
Tables[0].Rows[0], radice);
this.treeView1.Nodes.Add(radice);
this.treeView1.
ExpandAll();
}
private void Aggiungi_nodo
(DataRow riga_parente, TreeNode nodo_parente) {
foreach (DataRow dr in riga_parente.Get
ChildRows(ALBERO)) {
if ((string) dr[0] != "0") {
//altrimenti cicla
//sempre nel primo nodo !!!
string stringa=(string) dr[2]; //cattura il testo
TreeNode nuovo_nodo= new
TreeNode(stringa);
nodo_parente.Nodes.
Add(nuovo_nodo);
Aggiungi_nodo(dr, nuovo_nodo);
}
}
}
. . .
```

Per poter creare i vari rami che compongono l'albero abbiamo dovuto richiamare tutte le volte una stored procedure che, a fronte dell'id semilavorato inviato, estraeva i record relativi ai sotto-semilavorati e li ritornava sotto forma di DataSet al client. I vantaggi delle stored procedure sono legati al fatto che esse sono procedure memorizzate sul server del DB, e la prima volta che vengono eseguite vengono anche compilate così da risultare molto più veloci nelle chiamate successive. Inoltre, le stored procedure consentono di incapsulare lato-server le logiche strettamente legate all'elaborazione dei dati, in mo-

COS'È UNA RELAZIONE RIFLESSIVA ?

Una relazione riflessiva si ha quando le colonne chiave esterna contenenti i riferimenti e le colonne chiave primaria a cui viene fatto riferimento si trovano nella stessa tabella.

Esempio semplice (da vedere): Si può tuttavia anche avere un self join senza una relazione riflessiva: in questo caso, il join utilizzerà una condizione diversa dall'uguaglianza.

do da non doverle includere nel client: questo di sicuro riduce parecchio il traffico di rete aumentando notevolmente la produttività e l'efficienza dell'applicazione.
A questo punto, pertanto, è lecito domandarsi se non era forse meglio delegare alla stored procedure tutte le operazioni di elaborazione e costruzione dell'albero, in modo da consentire al client di lavorare direttamente con il resultset completo generato dal DB: ed è proprio questo aspetto che analizzeremo, finalmente, nel paragrafo successivo.

I VANTAGGI DELLE STORED PROCEDURE

Proviamo quindi a tradurre in Transact Sql il codice scritto in C# finora considerato. Anche in questo caso realizzeremo 2 semplici procedure: la prima costruisce il primo ramo dell'albero e chiama la seconda, la quale aggiunge di volta in volta i vari sottorami e completa lo sviluppo della gerarchia. La logica sarà, praticamente, la stessa di quella adottata inizialmente:

```
CREATE procedure Inizia_procedura
  @par_inizio as int
as
   set nocount on
    declare @description varchar(50)
  declare @level as int
 declare @tb_temp varchar(64)
 declare @stringa varchar(128)
begin transaction
          set @tb_temp =
          '##tabella_temporanea' +
                         convert( varchar(4), @@spid )
          set @stringa = 'create table '
          + @tb_temp
                         +' (id_materiale int not null,
                               descrizione varchar(50)
                        not null, parent_id int not null)'
          execute sp_executesql @stringa
          exec Ricor @par_inizio,  @tb_temp
          set @stringa = 'select * from '
          + @tb_temp
          execute sp_executesql @stringa
commit transaction
GO
CREATE procedure Processa_rami
  @par_inizio as int
as
   set nocount on
    declare @description varchar(50)
  declare @level as int
 declare @tb_temp varchar(64)
 declare @stringa varchar(128)
```

```
begin transaction
          set @tb_temp =
          '##tabella_temporanea' +
                         convert( varchar(4), @@spid )
          set @stringa =
           'create table ' + @tb_temp
                         +' (id_materiale int not null,
                               descrizione varchar(50)
                        not null, parent_id int not null)'
          execute sp_executesql @stringa
          exec Ricor @par_inizio,  @tb_temp
          set @stringa = 'select * from ' +
          @tb_temp
          execute sp_executesql @stringa
commit transaction
GO
```

Il problema principale delle stored procedure annidate è, tuttavìa, dovuto al limite fissato da Sql Server a max 32 livelli di annidamento, oltre i quali viene generato un errore di overflow.
Cercheremo pertanto una soluzione che sìa il più possibile universale, slegata da particolari limiti (eccetto quelli fisici della macchina host, ovviamente !!!), e consenta di modellare qualsiasi tipo di gerarchia in modo semplice ed elegante. Abbiamo visto che l'adozione delle funzioni ricorsive è molto semplice da implementare, tuttavìa essa presenta molteplici inconvenienti che quindi tendono a sconsigliarla. Il nostro obiettivo, pertanto, è quello di costruire un nostro personale "stack", costituito da una tabella temporanea dentro la quale vengono depositati tutti i vari rami da processare. Ogni ramo, appena giunto in tabella, viene "accatastato" e **immediatamente** elaborato: al termine dell'operazione, esso sarà subito **rimosso** per far posto ai successivi rami. Il principio di cui ci avvaliamo è, pertanto, quello stesso che utilizzano le funzioni in generale (e le funzioni ricorsive in particolare) per consentire la restituzione del controllo alla funzione chiamante (LIFO ->Last In, First Out). Una volta cancellato un ramo, la stored procedure risale lo "stack" passando a processare il ramo immediatamente successivo, quindi lo rimuove e riesegue la stessa procedura nei confronti degli altri elementi fino ad esaurire l'intera "pila".

CONCLUSIONI

L'uso delle relazioni riflessive nella progettazione di un database non è sicuramente consigliato. Tuttavia come nell'esempio dei semilavorati che abbiamo proposto rappresenta l'unica via d'uscita a problemi comlessi.

Enrico Viale

GENERAZIONE DI REPORT IN JAVA

IN QUESTO ARTICOLO IMPAREREMO A SFRUTTARE JASPER REPORTS PER GENERARE REPORT IN FORMATI QUALI: PDF, EXCEL, WORD, HTML. INOLTRE VEDREMO COME SIA POSSIBILE VISUALIZZARLI ALL'INTERNO DI UN BROWSER

Un report non è nient'altro che una visualizzazione "customizzata" di un insieme di dati, di solito recuperati da un database. In altre parole, i report sono utilizzati per estrarre un sottoinsieme di dati da un qualsiasi sistema di persistenza e visualizzare tali informazioni in formati gradevoli all'utente. Tali formati vanno da, ad esempio, un file PDF ad un grafico, quale potrebbe essere il diagramma a torta. In quest'articolo vedremo come utilizzare un prodotto open source, JasperReports, e il linguaggio Java per generare dei report in vari formati: dall'HTML al PDF, passando per Excel e Word.

SOFTWARE DI CUI ABBIAMO BISOGNO

I software da installare per provare gli esempi presentati in questo articolo sono:

- JasperReports
- iReport
- MySQL

Il primo software non è nient'altro che la libreria che utilizzeremo per generare i report dall'interno di codice Java. È un insieme di API interamente scritte con il linguaggio della Sun. Non è necessario scaricare tale libreria in quanto ce la ritroviamo sotto la cartella lib nella directory principale di iReport. Quest'ultimo è un tool che ci permette di "disegnare" i nostri report visivamente, ovvero utilizzando le classiche tecniche di drag&drop ecc. Se avete un sistema Windows potete scaricare l'installer altrimenti l'archivio fa al caso vostro. MySQL, invece, è il database che utilizzeremo come mezzo di persistenza dei dati. Ovviamente siete liberi di utilizzare qualsiasi server di database per provare gli esempi di questo articolo. Consiglio MySQL solo perché ha il "leggero" vantaggio di essere gratis!

ARCHITETTURA COMPLESSIVA

Arrivati a questo punto è opportuno chiarire come funziona la generazione dei report in JasperReports. Gli ingredienti per produrre un report sono:

- Un insieme di dati. Nel caso più comune, e anche nel nostro, un database.
- Un file in formato XML che rappresenta il design del report, cioè che ne descrive il suo layout. Per default questo file avrà estensione .jrxml.
- L'engine di JasperReports che mette assieme le due cose e produce in output il report vero e proprio nei suoi vari formati.

La figura 1 dovrebbe chiarire il concetto.

Fig. 1: *Architettura riguardante la generazione dei report*

IL NOSTRO DATABASE

Iniziamo col primo punto, ovvero con la costruzione del DB. Per semplificare notevol-

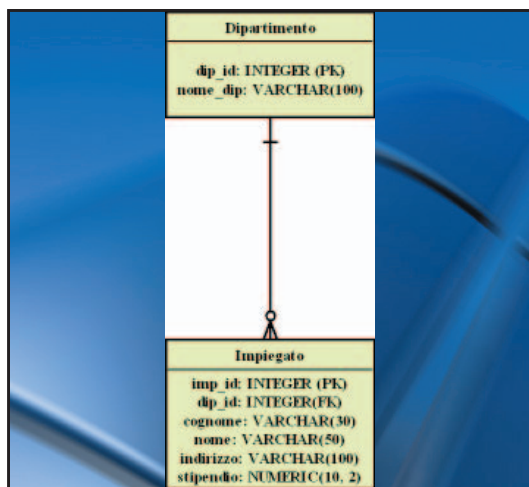



Fig. 2: **Schema del nostro DB**

mente il tutto la nostra base di dati sarà, volutamente, molto semplice. La figura 2 ne illustra lo schema.
Come si può vedere, vi sono solo due tabelle: Impiegato e Dipartimento. Per semplicità consideriamo che un impiegato possa lavorare in uno e un solo dipartimento, mentre in un dipartimento lavorano 0 (se, ad esempio, il dipartimento è appena stato fondato) o N impiegati. Vi è, quindi, una relazione 1 a N tra le due tabelle. Abbiamo risolto tale relazione introducendo la chiave primaria della tabella dipartimento all'interno della tabella impiegato. Per generare le tabelle, provviste già dei dati, create un DB di nome test e dopo utilizzate lo script che trovate assieme al codice allegato all'articolo.

IL DESIGN DEL REPORT IN FORMATO XML

Creare "a mano" il file XML che rappresenta il design del report, non è un compito piacevole. Per tale motivo utilizzeremo iReport che farà la maggior parte del lavoro per noi. Lanciamo iReport e creiamo un nuovo documento attraverso la voce di menu File/New Document; come nome del file usate dipendenti.jrxml. L'estensione .jrxml è lo standard che viene usato per indicare un report design di JasperReports in formato XML.
La schermata che ci viene presentata sarà simile a quella mostrata in figura 3.
Descriviamo, brevemente, le varie sezioni di un report:

- **Title**: Ne costituisce il titolo. Viene stampato solo all'inizio del report
- **Page Header**: A differenza del titolo questo elemento è ripetuto all'inizio di ogni pagina del report.
- **Column Header**: In questa sezione, vanno inserite le intestazioni delle colonne riguardanti i dati che vogliamo visualizzare. Nel nostro caso, ad esempio, avremo: Cognome, Nome, Indirizzo ecc.
- **Detail**: Questo è l'elemento più importante. In questa sezione, infatti, saranno inseriti i dati estratti dal nostro database.
- **Column Footer**: Come è facile intuire, questa sezione va alla fine di ogni colonna.
- **Page Footer**: Tale elemento viene visualizzato alla fine di ogni pagina costituente il report. Èuna buona idea inserire in questa sezione elementi quali il numero di pagina, la data e l'ora ecc.
- **Last Page Footer**: A differenza del precedente, gli elementi inseriti in questa sezione saranno visualizzati solo alla fine dell'ultima pagina del report.
- **Summary**: Si intuisce facilmente l'utilizzo di questa sezione, ovvero visualizzare dati riassuntivi del report subito l'ultima riga rappresentante i dati.

C'è da dire che nessuna di queste sezioni è obbligatoria. Possiamo liberamente utilizzare solo quelle che, per i nostri bisogni, hanno più senso. Ipotizziamo che le colonne del DB che vogliamo visualizzare siano: Cognome, Nome, Indirizzo e Stipendio di ogni impiegato, oltre al nome del Dipartimento in cui lavora. Utilizzando queste informazioni iniziamo con la progettazione visuale del nostro report utilizzando iReport. Il risultato che vogliamo ottenere è visibile in figura 4.
Come si può vedere le sezioni che utilizzeremo saranno: Title, Column Header e Detail. Cliccate col tasto destro del mouse su un

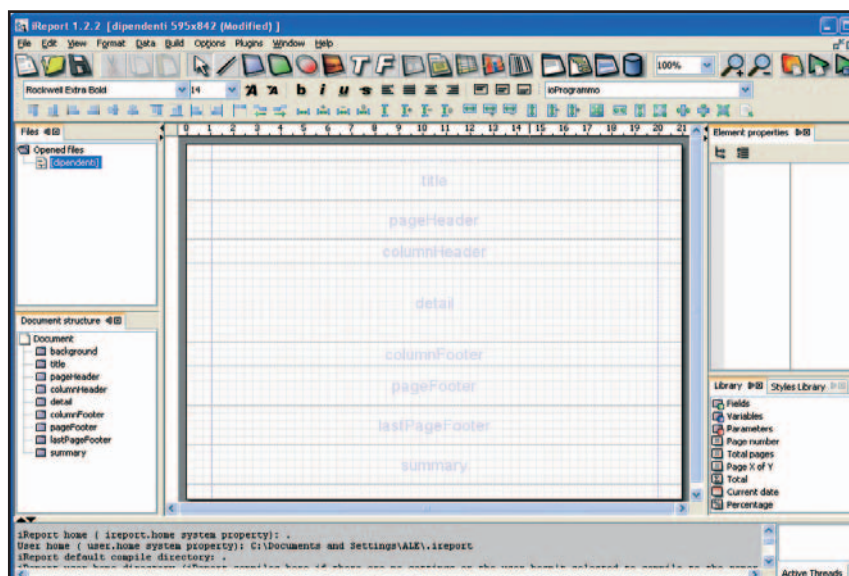



Fig. 3: **Screenshot principale di iReport**

Fig. 4: **Design del nostro report**

qualsiasi punto del report e selezionate Band properties. Per la sezione Title selezioniamo un'altezza di 50; per Column Header e Detail un'altezza di 20. Per tutte le altre sezioni un'altezza di 0 che le renderà invisibili. In Title inseriamo un oggetto di tipo Rounded Rectangle. Per inserire un oggetto all'interno del report potete procedere in due modi. Il primo consiste nell'utilizzare la voce di menu Edit/Insert element, mentre il secondo metodo è quello di cliccare sull'apposita icona presente nella toolbar sotto il menu. Ad esempio, nel caso del rettangolo con gli angoli arrotondati, possiamo inserirlo:

- Selezionando Edit/Insert element/ Rounded Rectangle.
- Alternativamente, cliccando sull'apposita icona visibile nella barra degli strumenti. A tal proposito, grazie ai tooltip, vi basterà soffermarvi su ciascuna icona per capire di quale oggetto si tratta.

Una volta posizionato il rettangolo selezioniamo uno Static text, inseriamolo all'interno del rettangolo ed editiamolo in modo da ottenere il titolo "REPORT DIPENDENTI". Ora aggiungiamo altri elementi di tipo Static text nella sezione Column Header cercando di riprodurre il risultato di figura 4. Gli elementi che, invece, appaiono nella sezione Detail sono i campi delle tabelle del DB da recuperare. Questo tipo di elementi sono dei Text Field e vanno inseriti in modo analogo agli altri tipi di oggetti. Una volta inseriti i Text Field accedete alle proprietà di questi ultimi e, sotto il tab Text Field inserite, in Textfield expression, l'espressione identificante il campo da visualizzare. Quest'ultima ha la seguente sintassi: $F{nome_campo}, dove nome_campo è il nome del campo selezionato nella query. Ad esempio, dato che la nostra query è:

```
SELECT cognome, nome, indirizzo, stipendio,
nome_dip as dipartimento
FROM impiegato i, dipartimento d
WHERE i.dip_id = d.dip_id
ORDER BY cognome
```

Fig. 5: **Impostazioni riguardanti il datasource in uso.**

i nostri Text Field saranno: $F{cognome}, $F{nome} e così via. Sarebbe impossibile coprire, in un solo articolo, tutte le proprietà che è possibile impostare per i report attraverso iReport. Giocateci un po' per capire come impostare sfondo, larghezza ecc.
La domanda che sorge spontanea a questo punto è: dove inserire la query riguardante il report?
Per rispondere dobbiamo, innanzitutto, vedere come specificare il Datasource in uso. Per fare ciò andate sotto Data/Connections /Datasources e selezionate New. Impostiamo le proprietà come illustrato in figura 5.
Ovviamente modificate i parametri mostrati in figura in base alle vostre impostazioni. Fatto ciò cliccate su Test per verificare che la connessione al DB sia attiva. Ricordatevi di avviare prima MySQL. Una volta testata la connessione, inserite la query nel report andando sotto Data/Report query. Qui, sotto

CREAZIONE DEL DB IN SEI PASSI

Creazione DB attraverso EasyPHP

Passo 1: Avviate EasyPHP.

Passo 2: Cliccate col tasto destro sull'icona presente nella Taskbar e lanciate PhpMyAdmin andando sotto la voce Configuration.

Passo 3: Create un nuovo database dandogli il nome test.

Passo 4: Arrivati a questo punto cliccate sulla voce SQL.

Passo 5: Aprite lo script che trovate assieme al codice allegato all'articolo e copiate ed incollate il codice all'interno dell'apposita finestra. Alternativamente potete indicare il percorso del file.

Passo 6: Cliccate su Go e, se tutto è andato bene, avrete il DB (compreso dei dati) pronto all'uso.

il tab Report query, “incolliamo” la nostra query e clicchiamo Ok.
A questo punto, se tutto è andato bene ed avete utilizzato lo script descritto al paragrafo precedente per la creazione del DB, dovreste essere in grado di produrre il vostro primo report direttamente da iReport. Per fare ciò, selezioniamo come file di preview il tipo PDF andando sotto Build/PDF preview (potete utilizzare, alternativamente, qualsiasi altro tipo di vostro gradimento).
Dopodiché scegliete Build/Execute (with active connection) dal menu o l’apposita icona dalla solita barra degli strumenti. Se tutto è andato a buon fine, dovreste ottenere un risultato simile a quello di figura 6.

USIAMO JASPER REPORTS DA CODICE JAVA

A questo punto sappiamo “disegnare” i nostri report e produrli sotto iReport. Tutto ciò, però, servirebbe a ben poco se non fossimo in grado di generare i nostri report direttamente attraverso codice Java. A tal proposito, ci vengono in aiuto le API della libreria JasperReports.
Per poter utilizzare queste API, dovete prima includere nel vostro progetto i jar necessari. Questi li trovate tutti sotto la cartella lib nella directory principale di iReport. Fatto ciò, vediamo come generare un report da Java.
Il primo passo da compiere è prendere il report design in formato XML e compilarlo. Il risultato sarà un file con estensione .jasper, il quale rappresenta un report design compilato. Il file in formato XML è quello con estensione .jrxml, ovvero quello che abbiamo generato utilizzando iReport.
Il codice necessario per compilare il file XML è il seguente:

```
public static JasperReport compileReport(String
            reportName, boolean compileXML) throws
                                    JRException
{
        JasperDesign jasperDesign = null;
        JasperReport jasperReport = null;
        if(compileXML ||
                !existJasperReport(reportName))
                                //compila il design
        {
                jasperDesign =
            JRXmlLoader.load(reportName + ".jrxml");
                jasperReport =
            JasperCompileManager.compileReport(
```

```
                                    jasperDesign);
        }
        else //carica il file .jasper
        {
                jasperReport =
                ReportLoader.loadReport(reportName);
        }
        //restituisce il file .jasper
        return jasperReport;
}
//restituisce true se esiste il file .jasper
private static boolean existJasperReport(String
                                    reportName)
{
        return new File(reportName +
                            ".jasper").exists();
}
```

Il metodo compileReport accetta due parametri. Il primo rappresenta il nome del file da compilare. Nel nostro caso, dato che avevamo chiamato il nostro file dipendenti.jrxml, passeremo come primo parametro la stringa “dipendenti”. Il secondo parametro indica se vogliamo forzare la compilazione del file .jrxml o meno. Infatti, una volta compilato il file .jrxml e ottenuto il file .jasper potremo utilizzare quest’ultimo evitando di compilare continuamente il report design. Il codice che carica e compila il file .jrxml è il seguente:

```
jasperDesign = JRXmlLoader.load(reportName +
                                    ".jrxml");
jasperReport =
JasperCompileManager.compileReport(jasperDesign);
```

Le classi JRXmlLoader e JasperCompile Manager fanno entrambe parte delle API di JasperReports. Per brevità non indicheremo il nome delle classi completo del package di

COGNOME	NOME	INDIRIZZO	STIPENDIO	DIPARTIMENTO
Catanese	Antonino	Via Felice Benlecco, 11 - Roma	30000	IT
De Gaetano	Domenico	Via Albert Einstein, 81 - Reggio Calabria	40000	Amministrazione
Ferraro	Antonio	Via Antonio Sisala, 25 - Reggio Calabria	30000	IT
Giardinello	Domenico	Via Leonardo Da Vinci, 123 - Reggio Calabria	38000	Logistica
Lacava	Michelangelo	Via Maria Tiammacchi, 72 - Reggio Calabria	40000	Amministrazione
Lacava	Alessandro	Via Francoise Pecorette, 69 - Milano	30000	IT
Morda'	Domenico	Via Beato Leccolano, 25 - Roma	30000	IT
Pitarelli	Luigi	Via Archi Cep, 69 - Bologna	32000	Edilizia
Scoglio	Vittorio	Via Sigmund Freud, 77 - Reggio Calabria	35000	Trasporti

Fig. 6: *Screenshot del file PDF rappresentante il nostro primo report*

appartenenza. Questo sarà evidente guardando il codice che si trova nel CD allegato.
Una volta compilato il report design, non ci resta che ottenere una connessione al DB in modo da recuperare i dati da utilizzare per riempire il report.
Fortunatamente le API di JasperReports ci permettono di passare i dati da utilizzare, per il riempimento del report, molto semplicemente. Ci basterà, infatti, passare un'istanza dell'oggetto java.sql.Connection ad un metodo di Jasper Reports e quest'ultimo si occuperà di fare il resto.
Questo perché, se ricordate, abbiamo già inserito nel report la query da utilizzare. Procediamo, quindi, con la generazione del report. Otteniamo una connessione al DB:

```
String DBURL = "jdbc:mysql://localhost/test";
java.sql.Connection conn = null;
[...]
conn = DBManager.getConnection(DBURL, "root",
                                          "");
```

DBManager è una classe che ho scritto per comodità. Il suo metodo getConnection restituisce un oggetto di tipo Connection. I parametri d'ingresso sono l'URL del DB, username e password.
Una volta ottenuta la connessione la diamo "in pasto" all'engine di JasperReports il quale, utilizzando un'oggetto di tipo JasperReport e l'oggetto Connection, costruisce un report con i dati, rappresentato da un oggetto JasperPrint.

```
JasperPrint jp = ReportFiller.fillReport(conn, null,
                                          jr);
```

DRIVER DI DATABASE PER MYSQL

Per poter utilizzar il DB MySQL da codice Java avete bisogno del driver adatto. Tale driver è Connector/J e lo potete liberamente scaricare andando al seguente indirizzo: http://dev.mysql.com/downloads/connector/j/3.1.html

JasperReports
Questa è la home del progetto. A questo indirizzo potete trovare le API e la documentazione di cui non potete fare a meno:
http://jasperreports.sourceforge.net/

EasyPHP
Se non siete pratici dell'installazione di MySQL allora potreste apprezzare EasyPHP. Installando quest'ultimo otterrete, in un colpo solo, Apache Web Server, MySQL, PHP e PHPMyAdmin. Quest'ultimo, in particolare, vi permetterà di creare il DB molto agevolmente attraverso una comoda interfaccia. L'indirizzo di riferimento è il seguente:
http://www.easyphp.org/?lang=it

MySQL
Tra i link non poteva mancare l'indirizzo del DB open source probabilmente più usato:
http://www.mysql.com/

Il codice del metodo fillReport è il seguente:

```
public static JasperPrint fillReport
    (Connection conn,Map params,JasperReport jr)
                                  throwsJRException
{
        JasperPrint jp = null;
        //riempie il report con i dati recuperati
                                              dalDB
jp = JasperFillManager.fillReport(jr, params,
                                              conn);
        return jp;
}
```

JasperFillManager è una classe di JasperReports. Nel nostro caso non stiamo utilizzando il parametro params il quale è un oggetto di tipo Map, infatti passiamo il valore null. Per ragioni di spazio non copriremo, approfonditamente, questa funzionalità di JasperReport. Vi basti sapere che è possibile definire dei parametri all'interno del report design e valorizzare tali parametri esternamente, proprio attraverso la Map. Un possibile parametro potrebbe essere, ad esempio, il nome dell'utente che ha lanciato la generazione del report. Passare tale parametro al report sarebbe semplicissimo:

```
Map params = new HashMap();
params.put("nomeUtente", "Alessandro Lacava");
```

Poi, per recuperare tale parametro dall'interno del report dovrete aggiungere un oggetto di tipo Text Field nella sezione desiderata (ad esempio nella Summary) ed inserire, in Textfield expression, l'espressione identificante il parametro. Tale espressione è simile a quella utilizzata per i campi del DB, se non per il fatto che al posto di $F viene utilizzato $P.
Nel nostro esempio: $P{nomeUtente}.
Non ci resta, a questo punto, che esportare il nostro report nel formato desiderato.

```
String REPORTS_FOLDER = "C:\\Program
 Files\\iReport-1.2.2\\MyReports\\ioProgrammo\\";
IReport reporter = new
            PdfReport(REPORTS_FOLDER+ "report");
reporter.exportReport(jp);
```

Tale codice esporta il report in un file in formato PDF di nome report.pdf. Ovviamente, dovrete modificare REPORTS_FOLDER a seconda delle vostre esigenze. L'interfaccia IReport è così codificata:

```
public interface IReport
```

```
{
        public void exportReport(JasperPrint jp)
                        throws JRException;
}
```

L'uso di tale interfaccia ci permetterà di scrivere diverse classi per esportare report in formati differenti (Excel, Word, ecc.). Ci basterà, infatti, implementare l'interfaccia appena vista ed utilizzarla come riferimento per la classe concreta (Strategy Pattern).

VISUALIZZARE UN REPORT ATTRAVERSO UN WEB BROWSER

Come preannunciato vedremo ora come visualizzare il report all'interno di un comune browser Web. Chiaramente, per questioni di spazio, non saranno analizzate tutte le impostazioni concernenti la creazione di servlet. Se non avete la necessità di utilizzare i report da Web, potete tranquillamente saltare questo paragrafo. Per esportare il report all'interno del Web browser ci basterà implementare opportunamente l'interfaccia creata in precedenza, ovvero IReport:

```
public class PdfToBrowserReport implements
                                    IReport
```

Quello che segue è il costruttore:

```
private HttpServletResponse response;
public PdfToBrowserReport(HttpServletResponse
                                    response)
{
        this.response = response;
}
```

Dopodiché occorre implementare l'unico metodo di IReport, ovvero exportReport:

```
public void exportReport(JasperPrint jp) throws
                                    JRException
{
        ServletOutputStream
                        outputStream = null;
        try
        {
                byte[] bytes = null;
                //esporta il report in un
                // array di byte
                bytes =
        JasperExportManager.exportReportToPdf(jp);
                //invia l'array di byte
                //all'output stream
                if (bytes != null && bytes.length > 0)
                {
response.setContentType("application/pdf");
response.setContentLength(bytes.length);
                        outputStream =
                        response.getOutputStream();
        outputStream.write(bytes, 0, bytes.length);
                        outputStream.flush();
                }
                else
                {
System.out.println("Errore nella generazione
                                del file pdf");
                }
        }
        catch(JRException jre)
    [...]
```

L'unica riga di codice degna di nota è JasperExportManager.exportReportToPdf(jp); il resto, infatti, dovrebbe essere noto ai programmatori che utilizzano servlet Java. Il metodo exportReportToPdf della classe JasperExportManager esporta l'oggetto JasperPrint in un array di byte.
Ricordiamo che l'oggetto JasperPrint rappresenta un report con all'interno i dati estratti dal DB. Istanziare la classe PdfToBrowserReport ed esportare il report dall'interno della servlet è semplice:

```
IReport reporter = new
            PdfToBrowserReport(response);
reporter.exportReport(jp);
```

Dove jp è un oggetto di tipo JasperPrint creato utilizzando la classe ReportFiller vista precedentemente e che troverete nel codice allegato.

CONCLUSIONI

In quest'articolo abbiamo visto come utilizzare JasperReports e Java per produrre report di livello professionale. L'utilizzo di report accurati fa parte di quel numero di caratteristiche che aggiungono valore al prodotto. Jasper Reports si configura come uno dei pochi prodotti per java in grado di produrre report accurati .Sicuramente torneremo a parlare di Jasper Reports e dei suoi aspetti avanzati.Nel frattempo vi abbiamo fornito le basi per poter produrre ottimi report, che potranno arricchire in modo le vostre applicazioni Java.

Alessandro Lacava

ALLA SCOPERTA DEL NUOVO IIS 7.0

MOLTO PIÙ CONFIGURABILE, RESTITUIRÀ IL CONTROLLO COMPLETO DELLE APPLICAZIONI ALL'UTENTE. ECCO QUALI SARANNO LE NOVITÀ CHE CARATTERIZZERANNO INTERNET INFORMATION SERVICE 7.0. VEDIAMO COME SFRUTTARLE

IIS 7.0 si preannuncia come uno dei componenti principali di Windows Vista e del prossimo Windows Server, che al momento è ancora conosciuto con il nome in codice di Longhorn.
Dalla sua parte ha senza dubbio il vantaggio di sfruttare al massimo quella che è stata la chiave di successo dalla concorrenza, cioè l'estrema componentizzazione, fermo restando le caratteristiche più importanti di IIS 6.0, cioè la **sicurezza** ed il forte supporto per il **.NET Framework**.
Il rilascio di questa nuova versione avverrà dunque insieme a Windows Vista, nel gennaio del 2007, ma già oggi è possibile, sin dalla beta 2, dare un'occhiata a quelle che sono le caratteristiche più importanti, così da trovarsi preparati quando sarà disponibile.

LA STRADA VERSO IIS 7.0

IIS 7.0 è un diretto discendente di IIS 6.0 per quanto riguarda l'architettura, che è sempre basata, all'incirca, sugli stessi concetti: ci sono ancora gli application pool, il servizio di monitoring dello stato di salute dei worker process e non è mutato il ruolo di http.sys, garanzia di una buona dose di performance nella ricezione ed invio dei dati.
IIS 6.0 ha l'invidiabile record, al momento, di avere a ben 3 anni dalla sua uscita **zero bug** di security noti, rappresentando una delle piattaforme più stabili per la messa in produzione di applicazioni web. Se ASP.NET ha riscosso tanto successo, specie nel segmento **enterprise**, buona parte dei meriti sono da dividere con IIS, che ha saputo garantire una stabilità ed un insieme di funzionalità che se paragonate ai predecessori, IIS 4 e 5, rendono più chiara la portata del cambiamento.
Se un limite va trovato ad IIS 6.0, che pure ha comunque migliorato la situazione rispetto ad IIS 5.0, è nelle modalità con la quale si configura il server web e nella sua scarsa estendibilità.
Nel caso specifico, infatti, IIS 6.0 è basato su un file di configurazione in formato XML, che è chiamato **metabase** e che nelle versioni precedenti era invece in formato binario.

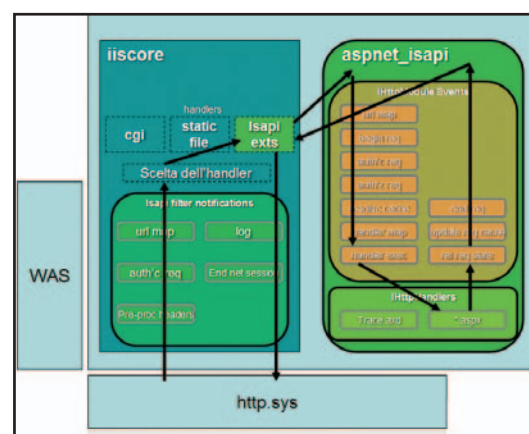



Fig. 1: ***L'architettura di IIS 6.0.***

Ovviamente l'utilizzo di XML come formato di delimitazione ha reso possibile una configurazione più semplice e la possibilità, in certi scenari, di portare la configurazione di tutti i siti su un nuovo server con una semplice operazione di copia.
Se c'è un'altra area dove IIS 6 pecca, sin dalle prime versioni, è quella della flessibilità di configurazione: è infatti praticamente impossibile demandare ad altri utenti la configurazione di un sito web, perché è tutta accentrata nelle mani dell'amministratore.
Se questa caratteristica è utile a preservare la sicurezza, perché ci sono impostazioni che è giusto che possano essere specificate solo da chi ha adeguati privilegi, mostra tutta la sua poca flessibilità in presenza di impostazioni "innocenti", come potrebbe essere l'impostazione di una pagina di errore.
Nel caso specifico, IIS 6.0 non consente di demandare questa impostazione al singolo

utente, richiedendo l'intervento di qualcunaltro, a meno che non si sfrutti ASP.NET, ma in questo caso viene del tutto saltato IIS, con il risultato di non avere comunque effetto sulle altre estensioni.
Dunque IIS 7.0 parte dal presupposto che debba essere garantita una **delegation** nella configurazione, attraverso meccanismi che consentano di definire al singolo sviluppatore le impostazioni locali delle proprie applicazioni, nel limite delle policy stabilite dall'amministratore.

CONFIGURAZIONE DI IIS CON IL WEB.CONFIG

Chi ha già utilizzato ASP.NET non avrà difficoltà a riconoscere il ruolo del web.config: è il file all'interno del quale viene specificata l'intera configurazione dell'applicazione, dalle stringhe di connessione fino alle impostazioni di protezione delle risorse.
IIS 7.0 abbraccia in toto questo concetto, aggiungendo una sezione denominata <***system.webServer*** />, all'interno della quale vengono ospitate tutte le configurazioni dell'applicazione.
In questa sezione trovano posto praticamente tutte le impostazioni che potrebbero essere fatte attraverso la classica console di gestione di IIS, che in effetti ora scrive direttamente dentro il file **applicationHosting.config**, che concettualmente ricorda da vicino il machine.config di ASP.NET.
All'interno di questo file sono definite le policy globali, così come eventuali application pool, siti web ed impostazioni di quest'ultimo.
Ovviamente il livello a cui vengono applicati è il medesimo: eventuali impostazioni locali sovrascrivono quelle dell'applicationHosting.config. Per rendere il tutto più facile da essere sfruttato dai sys admin che già hanno tool che lavorano con il metabase, esiste un servizio che tiene in sincronia questi due storage, garantendo che una modifica al file di config venga riflessa nel metabase e viceversa, per raggiungere dunque una compatibilità pari al 100% con l'attuale versione di IIS.
Il vantaggio di questo approccio basato sulla possibilità di stabilire le impostazioni nel web.config è soprattutto nella possibilità di poter definire le impostazioni del sito durante la fase di sviluppo e poi portarle, insieme a tutta l'applicazione, in produzione, senza dover toccare nient'altro.
Qualsiasi configurazione, dalle pagine di errore alle estensioni personalizzate, passando per il logging, le impostazione di security per inibire l'accesso ad alcuni IP, vengono copiate insieme al web.config, garantendo dunque che non sia necessario, quando si sposta un sito, ripeterle tutte.
È un vantaggio di non poco conto se si pensa che in questo modo è possibile **trasferire un sito semplicemente copiandolo,** piuttosto che un intero server web, per avere meccanismi di fault tolerance davvero facile da applicare.
Ad esempio, per rimuovere il supporto per le CGI, il config potrebbe contenere qualcosa del genere:

```
<configuration>
  <system.webServer>
    <serverFeatures>
      <remove name="Cgi" />
    </serverFeatures>
  </system.webServer>
</configuration>
```

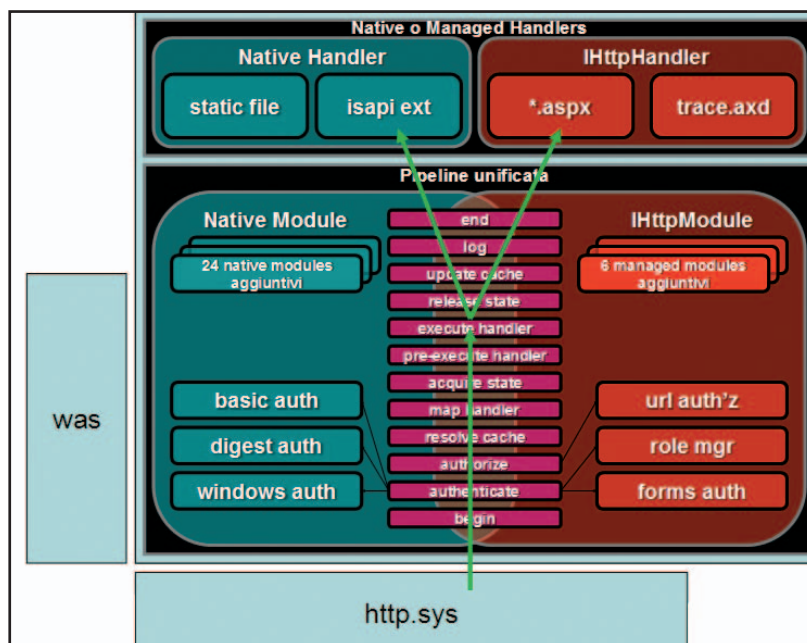



Fig. 2: *L'architettura unificata di IIS 7.0, con la pipeline integrata con quella di ASP.NET.*

Appare chiaro che essendo XML è praticamente possibile manipolarlo in qualsiasi

DIAGNOSTICA GRANULARE

Tutte le richieste sono visibili in real time, così che è possibile stabilire subito quale sia l'eventuale pezzo di codice che consuma CPU o memoria.

Si possono inoltre monitorare application pools ed applicazioni, così da tenere subito sott'occhio quale sono quelle che consumano maggiormente risorse.

modo, dato che basta un DOM per rendere questo possibile.
In realtà IIS 7.0 viene distribuito con un nuovo tool che è in grado di effettuare la gestione sia locale sia remota di un server, attraverso una nuova interfaccia che altro non fa che agire su questo file in formato XML, più un insieme di nuovi tool da riga di comando ed i soliti script WMI, che continuano ad essere compatibili con quelli di IIS 6.0.
L'utilizzo dell'interfaccia grafica è davvero

***Fig. 3:** I moduli già inclusi in IIS 7.0. Possono essere aggiunti o rimossi liberamente.*

semplice e consente di modificare qualsiasi sezione sia stata registrata. È in più possibile creare dei propri moduli, per estenderne le funzionalità, così da poter offrire per eventuali componenti custom anche una parte di gestione visuale, direttamente integrata all'interno del tool di gestione di IIS.

UNA NUOVA PIPELINE INTEGRATA

IIS 7.0 integra nativamente il supporto al mondo **managed**, cioè quello basato sul .NET Framework. Per farlo, la pipeline di richiesta e risposta è stata riscritta, in modo da poter aggiungere anche moduli basati sul .NET Framework.
L'unica via in IIS 6.0 per modificare qualcosa a livello globale consiste nella creazione di un **filtro ISAPI**, che è generalmente scritto in C++. Questo filtro va installato in modo tale che IIS passi attraverso il codice che contiene, per poi girare la richiesta all'effettivo runtime (ASP, ASP.NET, PHP o CGI) che si occuperà di elaborare e fornire un risultato. È dunque praticamente impossibile avere un buon grado di flessibilità, dato che i filtri ISAPI sono oggetti COM, che vanno registrati nel sistema, soffrono del versioning e di tutto quello che COM ha come limite.
L'idea alla base di IIS 7.0 è invece quella di fornire una pipeline che integri all'interno della stessa sia moduli managed sia nativi, unificandola.
Questo concetto è già implementato in ASP.NET, sin dalla prima versione, attraverso gli **HttpModule**. Si tratta di classi particolari che si registrano per alcuni eventi dell'applicazione e vengono invocate per eseguire il codice associato.
Gli HttpModule sono molto comodi poiché consentono di aggiungere funzionalità dall'alto, senza cioè toccare le singole pagine dell'applicazione: agiscono in automatico senza che sia necessario fare nient'altro che registrarli, all'interno del web.config.
IIS 7.0 porta questo modello all'interno del server web stesso, con il vantaggio che un modulo managed può fare le stesse identiche cose di uno nativo, quindi agire anche su altre tipologie di risorse, come le pagine ASP. La maggior parte delle funzionalità chiave di ASP.NET sono implementate proprio mediamente HttpModule: **OutputCache** o **FormsAuthentication**, piuttosto che **UrlAuthorization**, per dirne alcune, funzionano proprio in questo modo.
Appare dunque chiaro che con IIS 7.0 è ora possibile implementare queste funzionalità in managed code, che è certamente più semplice da creare e gestire di un filtro ISAPI, con il vantaggio che saranno richiamate anche nel caso di altri motori di scripting: il tipico esempio è quello che consente di proteggere una **pagina PHP** sfruttando la Forms Authentication. Con IIS 6.0 non è impossibile, ma certamente non è così semplice.
Per farlo con IIS 7.0 è necessario installare l'ultima versione di PHP sotto IIS 7.0, poi procedere alla definizione delle policy di autenticazione ed autorizzazione, così come si farebbe solo per ASP.NET, ad esempio sfruttando le Membership API. Il discorso vale chiaramente anche per ASP o qualsiasi altra estensione.
A questo punto è sufficiente richiamare la pagina PHP per avere la stessa protezione che si avrebbe con una ASP.NET. Ed il discorso vale in generale per qualsiasi risorsa, anche pagine ASP, immagini, ZIP o file PDF.
Tutto ciò è possibile poiché in IIS 7.0 sono presenti due tipi di application pool, a seconda della tipologia di applicazioni. Il primo è il classico tipo di application pool a cui siamo abituati con IIS 6.0. Prende il nome di **ISAPI Pipeline mode** e non prevede il supporto per

moduli managed, cosa che invece fa l'**Integrated Pipeline mode**, che in pratica fonde il runtime di ASP.NET dentro la richiesta, tanto che cambia radicalmente il modo in cui è gestita la stessa, con benefici anche dal punto di vista delle performance. In quest'ultima modalità, infatti, la pipeline passa una volta sola attraverso gli eventi, e sia i moduli managed sia quelli nativi agiscono in contemporanea in questa fase, laddove nella modalità ISAPI, nel caso di applicazioni ASP.NET, i moduli managed vengono invocati dopo che quelli nativi hanno già avuto effetto.
Dunque in modalità Integrated, come il termine stesso suggerisce, l'integrazione con ASP.NET è totale e porta innumerevoli benefici.

CREARE MODULI PER IIS 7.0

IIS 7.0 è distribuito già con tantissimi module, che hanno praticamente gli usi più disparati, dalla gestione della pagina di default agli errori, passando per l'autenticazione Windows, quella Basic, piuttosto che il directory browsing.
Tutti questi componenti sono registrati globalmente nell'applicationHosting.config e possono essere rimossi, se la sezione viene sbloccata, anche nei web.config locali.
L'effetto è quello di poter creare una minor superficie di attacco, rimuovendo i moduli che non sono strettamente necessari, ma anche quella di aggiungere moduli managed, se l'application pool è in Integrated mode.
In quest'ultimo caso, è necessario creare un normale e banale ***HttpModule***, cioè una classe che implementi l'interfaccia ***IHttpModule*** del namespace ***System.Web***.
Come esempio potremmo prendere un semplice HttpModule che aggiunga la data ed ora attuale ad ogni singola pagina, come questo:

```
public class DateTimeModule :
                    System.Web.IHttpModule
{
void Init(HttpApplication context)
{
context.BeginRequest += new
            EventHandler(this.BeginRequest);
}
void BeginRequest(Object source, EventArgs e)
{
HttpApplication application =
                 (HttpApplication)source;
```

```
HttpContext context = application.Context;
context.Response.Write(DateTime.Now());
}
void Dispose(){ }
}
```

Dopo averlo creato, è sufficiente registrarlo nella sezione modules, in questo modo:

```
<configuration>
<system.webServer>
<modules>
<add name="DateTimeModule"
                 type="DateTimeModule" />
</modules>
</system.webServer>
</configuration>
```

Fatto questo, per ogni richiesta ad una pagina si avrà ora la data ed ora stampata. Ovviamente gli utilizzi che si possono fare di questo approccio sono i più disparati e sicuramente più interessanti di quello mostrato, tuttavia è indicativo di quello che IIS 7.0 promette di rendere possibile.
Per rendere performante il tutto, è stata fatta la scelta conservativa di non far passare tutti i module per l'Integrated Pipeline, per cui è necessario specificarlo aggiungendo l'attributo preCondition="integratedMode" nel nodo <add />, in fase di registrazione dello stesso.
Resta inoltre possibile, senza alcuna limitazione, la creazione di moduli nativi, come filtri ISAPI.

CONFIGURARE IIS 7.0

Una volta chiaro che IIS 7.0 è configurabile attraverso file XML, diventa semplice iniziare a modificarne le proprietà. Come detto è possibile farlo anche attraverso il tool di gestione visuale, che altro non fa che agire sul file applicationHosting.config.
Volendolo fare manualmente, lo schema è comunque semplice e non rappresenta particolari difficoltà. Ad esempio per modificare le pagine di default, basta utilizzare questa sintassi:

```
<configuration>
    <system.webServer>
        <defaultDocument>
            <files>
                <add value="default.aspitalia" />
            </files>
        </defaultDocument>
    <system.webServer>
```

```
</configuration>
```

Deve esistere, ovviamente, una pagina con quel nome nella directory richiesta, così da fare in modo che se non dovesse essere specificata uan pagina, l'utente veda comunque il contenuto di quella di default. Per associare l'estensione personalizzata, è sufficiente cercare la sezione <handlers /> e copiare, ad esempio, la configurazione per .aspx.
Si può aggiungere praticamente qualsiasi tipo di impostazione, come un'header alla risposta, così:

```
<httpProtocol>
<customHeaders>
<add name="X-WebSiteUrl"
        value="http://www.aspitalia.com/" />
</customHeaders>
</httpProtocol>
```

Fig. 5: **La gestione visuale della delegation della configurazione.**

Molto comoda la possibilità di definire una pagina di errore globale, non solo per le applicazioni ASP.NET:

TRACING DELLE RICHIESTE

È possibile tracciare l'intera vita della richiesta, così da verificare in quale punto mancano i permessi o si verifica un errore. Da questo punto di vista, esiste un meccanismo di tracing automatico per tutte le richieste con errori, così come uno specifico per quelle che vanno in timeout e ci mettono troppo tempo per completare, con un errore del tipo "takes too long". Ovviamente tutte queste impostazioni sono configurabili in base alla singola applicazione. Grazie alla pipeline unificata, infine, gli eventi di ASP.NET e di IIS vanno a finire nello stesso trace log, con la possibilità di accedere agli stessi nello stesso identico modo.

```
<httpErrors>
<error statusCode="404" url="/errors/404.aspx"
                                             />
</httpErrors>
```

Più in generale, attraverso la sezione <system.webServer />, è possibile modificare qualsiasi impostazione del server web.

DELEGATION NELLA CONFIGURAZIONE

Un concetto fondamentale per rendere IIS 7.0 l'ambiente ideale anche per i fornitori di hosting condiviso è la possibilità di delegare, da parte dell'amministratore, parte della configurazione allo sviluppatore, cioè la persona che ha fisicamente accesso al web.config.
Per fare questo IIS 7.0 utilizza, in tutte le sezioni di configurazione, il concetto di **delegation**. In fase di installazione, almeno al momento, gran parte delle sezioni sono bloccate ed è possibile configurarle solo attraverso l'applicationHosting.config.
Le ragioni di questa scelta conservativa sono da ricercarsi nella necessità di garantire che, di default, ci sia la minor superficie possibile di attacco.
Ogni nodo può essere delegato, e quindi modificato nel web.config, almeno attraverso 3 modalità differenti.
La più semplice di tutte è ovviamente l'utilizzo dell'interfaccia di amministrazione, seguita a ruota dalla modifica del file di config e da un eseguibile da riga di comando.
Nel secondo caso, è possibile modificare un nodo aggiungendo l'attributo allowOverride, che può assumere il valore true se questa modifica deve essere ammessa nei config figli, false se invece deve essere bloccata.

```
<authorization allowOverride="true" />
```

È anche possibile specificare un'impostazione da ereditare nei config figli, semplicemente aggiungendo l'attributo *inheritInChild Applications= "true"*.
Operazione analoga è possibile lanciando da riga di comando qualcosa come:

```
%systemroot%\system32\inetsrv\appcmd unlock
                      config /section:authorization
```

Ovviamente questa tecnica può essere utilizzata anche per inibire la modifica di alcune sezioni, così da rendere più sicuro in certi contesti il funzionamento del server web.

IIS 7.0 NON SOLO HTTP

IIS 7.0 nasce con l'esigenza di fornire un'architettura tale da garantire l'hosting non soltanto di applicazioni web. Potremmo dire che IIS 7.0 rappresenta un sistema generalizzato per l'attivazione di processi e l'health monitoring (cioè, il controllo della salute) degli stessi. Il tutto è possibile grazie a **WAS** (Windows Activation Service), un componente che già in IIS 6.0 si occupava di garantire che gli application pool girassero senza problemi e che ora è stato esteso per supportare un modello generico per applicazioni distribuite. IIS 7.0 ha già infatti incluso il supporto per HTTP, NET.TCP, NET.PIPE e NET.MSMQ.
Questo vuol dire che è in grado di far girare applicazioni HTTP e non-HTTP side-by-side, cioè contemporaneamente, al punto tale che ASP.NET e WAS possono utilizzare lo stesso application domain, nello stesso worker process. Il motivo di questa architettura è quello di favorire un ambiente nel quale gestire i servizi di **Windows Communication Foundation**, un framework unificato per la costruzione di applicazioni service-oriented, cioè fortemente orientate ai servizi, sulla piattaforma Windows, che sarà disponibile con il **.NET Framework 3.0**. Conosciuto anche come WinFX, sarà distribuito proprio insieme a Windows Vista, ma anche per Windows XP e Windows Server 2003.
Questa architettura rende possibile la comunicazione intra-processo per tipi di applicazioni differenti. Il vantaggio principale è che i servizi WCF posso utilizzare funzionalità esistenti in ASP.NET, come State Service, Globalization, Membership, Roles o Profile.
In più, questo rende possibile un modello unificato ed unico per sviluppo, deployment e gestione delle soluzioni, con il vantaggio che non è necessario apprendere nuove tecniche, né per lo sviluppatore, né per il sistemista.
L'architettura di IIS 6.0, come detto, prevede la presenza di un sistema di monitoraggio dei processi, che è chiamato WAS e tra le altre cose serve anche per riciclare gli stessi qualora ci siano problemi.
In IIS 6.0 questo componente è dentro **W3SVC.exe**, che si occupa di fare l'hosting vero e proprio dei worker process, come quello ASP.NET, e lavora insieme ad HTTP.sys nel core del web server, per rispondere alle richieste che arrivano attraverso questo protocollo.
In IIS 7.0 WAS garantisce che ci sia sempre un ottimo controllo dello stato dei processi, grazie ad una forte componentizzazione dell'architettura, che fa sì che WAS sia un processo esterno rispetto ad IIS stesso.
W3SVC è stato infatti diviso in due componenti principali, WAS e W3SCV, con il risultato che ora WAS può essere ora utilizzato anche per scenari non-HTTP, come nel caso di servizi WCF, che lavora anche su listener TCP piuttosto che su code. Con questa architettura, WAS è ora in grado di funzionare su TCP, Named Pipes, MSMQ e HTTP, garantendo le stesse funzionalità sia ad IIS 7.0 sia ai servizi di WCF.

Fig. 6: L'associazione delle estensioni al relativo gestore

CONCLUSIONI

Ci sarebbero tante cose da dire su IIS 7.0, ma questa anteprima dovrebbe darvi la giusta misura rispetto a quello che è possibile fare migliorando ulteriormente quello che già IIS 6.0 ha mostrato di saper offrire in questi anni. IIS 7.0 migliorerà di sicuro la sicurezza intrinseca, grazie alla forte componentizzazione, ma sarà senza dubbio un ottimo strumento soprattutto per via della sua straordinaria flessibilità, che consentirà a tutti di creare componenti aggiuntivi in maniera molto più immediata. Non ci resta che aspettare Gennaio 2007, con Windows Vista prima, e la fine dello stesso anno, per Windows Server Longhorn.

Daniele Bochicchio

APPROFONDIRE IIS 7.0

IIS 7.0 ha già diversa documentazione, articoli ed esempi pronti.
Si può partire dal sito ufficiale, raggiungibile su
http://www.iis.net/default.aspx?tabid=7

La documentazione, in beta, è invece disponibile su
http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/iis7/Ops/712143da-96b1-4afb-8ea4-615a284c7e7a.mspx?mfr=true

SOFTWARE E CATENE DI RESPONSABILITÀ

I PRINCIPI DELLA PROGRAMMAZIONE OBJECT ORIENTED PREVEDONO CHE OGNI OGGETTO PENSI UNICAMENTE A SÉ STESSO. VI SPIEGHIAMO PERCHÉ QUESTO È GIUSTO E COME SIA POSSIBILE IMPLEMENTARE CLASSI INDIPENDENTI FRA LORO

Chi ha studiato, o sta studiando, le proprietà che il codice deve avere per essere considerato "di qualità" si sarà di certo imbattuto nella definizione di "accoppiamento" (coupling). Lungi dalla volontà di chi scrive l'idea di incentrare questo articolo sui rituali amorosi di una qualche specie animale esotica chiariamo immediatamente che l'accoppiamento è una di quelle proprietà "negative", che un codice di qualità dovrebbe cercare di non possedere.
L'accoppiamento indica in quale misura un oggetto di una classe abbia bisogno di un oggetto di un'altra classe, e solo di quell'oggetto, per potere svolgere il proprio lavoro. Uno dei principi fondamentali della programmazione a oggetti è quello di isolare i concetti, assegnando a ogni classe uno specifico compito, che sia slegato il più possibile dal modo in cui operano oggetti di altre classi.
Il pattern che illustreremo di seguito è detto Chain of Resposibility (Catena di Responsabilità) e il suo scopo è proprio quello di evitare l'accoppiamento tra l'oggetto che genera una richiesta e l'oggetto che deve soddisfare questa richiesta. Questo è ottenuto dando la possibilità a più oggetti di analizzare ed eventualmente soddisfare una richiesta, in maniera del tutto trasparente al richiedente, ovvero senza che il richiedente debba sapere quale oggetto è in grado di manipolare correttamente la sua richiesta.

IL PROBLEMA

Partiamo come di consueto da un esempio del mondo reale per cercare di arrivare a una giustificazione intuitiva della bontà del pattern.
Supponiamo di dovere realizzare un software per la gestione degli acquisti da parte di un negozio on-line. Poniamo che il negozio accetti ordini di tipi molto diversi tra loro. Ad esempio:

- ordini da aziende
- ordini da privati cittadini
- ordini di grosse quantità di prodotti
- ordini di prodotti che non sono ancora sul mercato
- ecc.

Ovviamente si vuole che questi ordini vengano trattati in maniera differente, ad esempio concedendo sconti su grosse quantità, sconti ai clienti più affezionati ecc. Il termine col quale ci si riferisce a queste politiche di trattamento degli ordini è detto "business logic". Naturalmente la business logic va di pari passo col business ed è variabile, per adattarsi ai desideri dei clienti. Ad esempio potremmo volere cambiare la logica con la quale viene scelto il corriere per le spedizioni, in base agli sconti offerti alla nostra società dal corriere stesso, oppure potremmo decidere, in occasione di un importante evento sportivo, di regalare una TV a schermo piatto ai nostri clienti qualora la squadra del cuore dovesse vincere, sperando in questo modo di fare ottimi affari. Qualsiasi sia la politica adottata per la business logic, il nostro codice dovrebbe essere abbastanza flessibile da potere implementare in maniera veloce e indolore tutti i cambiamenti richiesti, senza alterare la sua struttura.
È ovvio quindi che l'idea che ci era balenata in mente di implementare un gigantesco e monolitico algoritmo che preveda tutti i casi possibili, simile a:

```
if (cliente_affezionato) {
...
}
else if (grande_quantita) {
...
}
...
else if (campioni_del_mondo) {
...
}
```

```
else ...
```

risulti essere del tutto inapplicabile. Da una parte infatti risulterebbe arduo intervenire sul codice per apportare una qualche modifica, seppur minima. Dall'altra ogni intervento andrebbe a minare la struttura stessa dell'intero procedimento decisionale, magari intaccando parti che non sono interessate dalla modifica stessa. Inoltre codificare algoritmi monolitici non è esattamente ciò che si intende per "architettura a oggetti". Vediamo come procedere in maniera più efficace.

LA SOLUZIONE

Quello che ci piacerebbe avere, nella soluzione ideale, sono alcune caratteristiche molto interessanti, ad esempio:

- possibilità di incapsulare all'interno di un oggetto tutta la logica per un certo tipo di trattamento della richiesta
- mantenere una struttura coerente e indipendente dagli oggetti che manipolano le richieste
- possibilità di richiedere la manipolazione di una richiesta senza conoscere quale sia l'oggetto che la soddisferà
- possibilità di modificare la business logic in maniera semplice, addirittura a run-time

Il pattern che ci offre una soluzione con caratteristiche molto vicine a quelle desiderate è proprio le chain of responsibility. Il concetto su cui si basa questo pattern è quello del disaccoppiamento tra generatore della richiesta e manipolatore della richiesta, ovvero l'oggetto che soddisferà in qualche modo la richiesta stessa. Quello che si fa, in pratica, è creare una lista collegata di manipolatori, ciascuno dei quali contiene il riferimento a un manipolatore successivo. Quando l'oggetto richiedente effettua la sua richiesta, ottiene un riferimento al primo manipolatore e gli passa un oggetto contenente gli estremi della richiesta stessa.
Il manipolatore (handler) analizza l'oggetto-richiesta e:

- **se** è in grado di soddisfare la richiesta, allora lo fa e restituisce il risultato desiderato
- **altrimenti** passa l'oggetto-richiesta al manipolatore successivo, cedendo la responsabilità di tutta l'elaborazione

È evidente che se almeno un manipolatore è in grado di soddisfare la richiesta si otterrà certamente il risultato cercato, questo proprio grazie al fatto che i manipolatori formano una catena e sono interrogati uno di seguito all'altro. Il diagramma UML del pattern in questione è visibile in FIGURA.

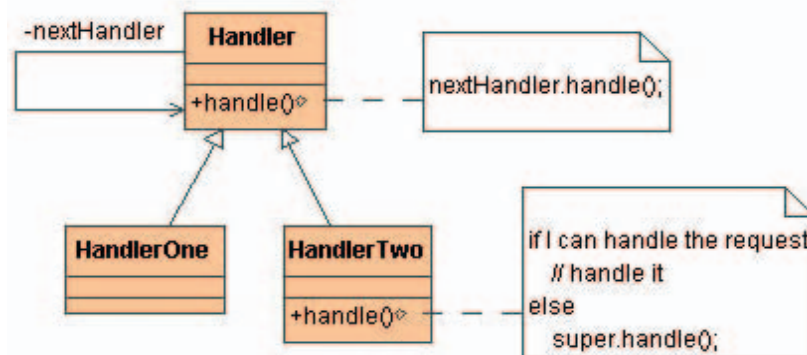



FIG.1: Il diagramma uml del pattern chain of responsability

Possiamo notare come tutte le caratteristiche che auspicavamo in precedenza siano presenti nella soluzione proposta. In particolare:

- il modo con cui ciascuna richiesta può essere manipolata è racchiuso in un unico oggetto: il manipolatore. Ciascun manipolatore è indipendente dagli altri
- la struttura a catena è mantenuta qualsiasi sia il numero dei manipolatori e qualsiasi sia l'implementazione specifica di ciascuno di essi
- il richiedente non ha bisogno di conoscere quale manipolatore è in grado di soddisfare la richiesta: tutto ciò che fa è ottenere il riferimento al primo anello della catena e

PICCOLA BIBLIOGRAFIA SUI PATTERN

Il libro che ha contribuito alla formalizzazione del concetto di "pattern" e che rappresenta il capostipite di questa letteratura è sicuramente quello citato nel corpo dell'articolo e cioè:

- **"Design Patterns: Elements of Reusable Object-Oriented Software" (Gamma, Helm, Johnson, Vlissides).**

Questo libro tuttavia, oltre a definire dei pattern, definisce anche una maniera standard per formalizzare un pattern. Anche grazie a questo fatto sono stati scritti diversi altri libri sui pattern, applicati in ambiti differenti. Ad esempio segnaliamo:

- **"Design Patterns Java Workbook" (Steven John Metsker)**
- **"Pattern Hatching: Design Patterns Applied" (John Vlissides)**
- **"Framework-based Software Development in C++" (Gregory F. Rogers)**
- **"Patterns and Skeletons for Parallel and Distributed Computing" (F.A. Rabhi and S. Gorlatch)**
- **"Small Memory Software: Patterns for Systems with Limited Memory" (James Noble & Charles Weir)**

"iniettare" l'oggetto-richiesta, in attesa del risultato

- è possibile modificare facilmente la business logic semplicemente implementando un nuovo manipolatore e aggiungendolo alla catena.

IL CODICE

L'implementazione di questo pattern prevede la codifica di una classe base astratta "manipolatore". Le funzionalità che ciascun oggetto derivato dovrà implementare per diventare manipolatore saranno relative ai seguenti compiti:

1. E' possibile manipolare la richiesta?
2. Manipola la richiesta
3. Ottieni riferimento al successivo manipolatore
4. Inserisci il riferimento al successivo manipolatore

Per tornare al nostro esempio di gestione degli ordini di un negozio, proponiamo il codice creato da Jeremy Miller, che risulta molto esplicativo in questo caso.
L'oggetto-richiesta è l'ordine effettuato dal cliente ed è chiamato Order:

```
public class Order
{
        private string _countryCode;
        private double _orderAmount;
        private double _customer;

        public string CountryCode
        {
            get { return _countryCode; }
            set { _countryCode = value; }
        }

        public double OrderAmount
        {
            get { return _orderAmount; }
            set { _orderAmount = value; }
        }

        public double Customer
        {
            get { return _customer; }
            set { _customer = value; }
        }
}
```

Come si vede sono in esso contenuti tutti i dati ritenuti significativi per un ordine, ad esempio l'ID del cliente che l'ha effettuato, la quantità di oggetti ordinati ecc. Su questi dati si baserà il manipolatore di turno per decidere se è in grado di soddisfare l'ordine oppure dovrà passare l'oggetto-richiesta Order al manipolatore successivo della lista collegata.
La classe base astratta che implementa il manipolatore generico, ovvero una sorta di interfaccia per le specifiche implementazioni, è la seguente:

```
public abstract class OrderHandler
{
        private OrderHandler _nextHandler;

        public void ProcessOrder(Order order)
        {
            if (this.CanHandle(order))
            {
                this.handleRequest(order);
            }
            else
            {
              _nextHandler.ProcessOrder(order);
            }
        }
        public abstract bool CanHandle
                          (Order order);
        protected abstract void
                handleRequest(Order order);
        public void AppendSibling(OrderHandler
                                   lastHandler)
        {
            if (_nextHandler == null)
            {
                _nextHandler = lastHandler;
            }
            else
            {
_nextHandler.AppendSibling(lastHandler);
            }
        }
}
```

Le funzioni astratte, ovvero le funzioni la cui implementazione è demandata agli oggetti derivati, sono:

- CanHandle() : ci dice se il manipolatore è in grado di soddisfare la richiesta
- handleRequest() : processa l'ordine

questi due metodi hanno il compito di incapsulare nell'oggetto in esame la business logic del manipolatore. Qui e soltanto qui è racchiusa la peculiarità di un manipolatore rispetto a un altro.
Le funzioni che sono implementate nella classe

base sono:

- ProcessOrder() : si occupa di processare la richiesta relativa a un Order. L'elaborazione avviene chiamando la handleRequest() locale se CanHandle() ha restituito un valore true, chiamando la handleRequest() del manipolatore successivo nella catena in caso contrario. È evidente la ricorsività di questo meccanismo che si ferma quando la richiesta è soddisfatta oppure quando non c'è nessun manipolatore in grado di soddisfarla.
- AppendSibling() : è utilizzata per aggiungere il riferimento al successivo manipolatore

Da notare la presenza di un membro privato della classe, nextHandler, che contiene appunto il riferimento al manipolatore successivo.
Le implementazioni dei singoli manipolatori dovranno derivare da OrderHandler:

```
public class MostFavoredCompanyHandler :
                              OrderHandler{...}
public class EuropeanOrderHandler :
                              OrderHandler{...}
public class LargeOrderHandler : OrderHandler{...}
public class DefaultOrderHandler :
                              OrderHandler{...}
```

Come si può vedere ciascun manipolatore svolge un compito specifico, come ad esempio gestire le richieste per i grossi ordini, per gli ordini provenienti da una determinata regione geografica ecc.
La classe che si occupa di utilizzare effettivamente tutto questo meccanismo viene chiamata nel nostro esempio OrderService:

```
public class OrderService
{
        public void ReceiveOrder(Order order)
        {
            OrderHandler handler =
                          createHandlerChain();
            handler.ProcessOrder(order);
        }
        private OrderHandler createHandlerChain()
        {
            OrderHandler topHandler = new
                           LargeOrderHandler();
            topHandler.AppendSibling(new
                       EuropeanOrderHandler());
            topHandler.AppendSibling(new
                          LargeOrderHandler());
            topHandler.AppendSibling(new
                  MostFavoredCompanyHandler());
            topHandler.AppendSibling(new
```

```
                       DefaultOrderHandler());
            return topHandler;
        }
}
```

La funzione pubblica ReceiveOrder() si occupa di creare la catena di manipolatori che processerà l'oggetto-richiesta e di iniettare la richiesta nella catena. Da notare come venga richiamata la ProcessOrder() su un generico oggetto OrderHandler. Il richiedente non sa di quale tipo sarà il primo manipolatore della catena, sa solo che esso può processare un ordine. La creazione della catena vera e propria avviene utilizzando createHandler Chain() e la funzione AppendSibling di OrderHandler.
Una possibile implementazione alternativa potrebbe essere l'utilizzo di un array di manipolatori, anziché una lista collegata. Questo renderebbe più semplice l'inserimento e la cancellazione di manipolatori dalla catena. In un software reale una ulteriore alternativa potrebbe essere quella di utilizzare un generico contenitore di manipolatori, da scorrere con un iteratore. Questo consentirebbe di astrarre il meccanismo dal particolare tipo di struttura utilizzata per raggruppare i manipolatori e potrebbe risultare più flessibile in pratica, nonché più elegante da un punto di vista esclusivamente architetturale.

CONCLUSIONI

In questo articolo abbiamo trattato le problematiche, le soluzioni e una implementazione, relative al pattern "chain of responsibility". I principali attori di questo pattern sono:

- una classe base astratta che racchiude il concetto di manipolatore di una richiesta
- una serie di classi derivate che incapsulano la business logic che soddisfi ciascuna richiesta
- un oggetto-richiesta valido per qualsiasi manipolatore
- un oggetto-richiedente che si occupa di costruire la catena e inoltrare la richiesta

L'utilizzo di questo pattern permette di conseguire notevoli risultati nel disaccoppiamento tra classi, realizzando in particolare una struttura in cui il richiedente non ha bisogno di sapere quale sia il manipolatore adatto alla sua richiesta e modificare il numero/tipo di manipolatori sia estremamente semplice.

Alfredo Marroccelli

SOFTWARE SUL CD

JBuilder 2005 Foundation

LA SOLUZIONE BORLAND PER LO SVILUPPO JAVA

Ha dell'incredibile questo IDE targato Borland. E' probabilmente l'unico vero ambiente RAD per lo sviluppo di applicazioni Java. Il suo concorrente potrebbe essere NetBeans ma per velocità e ricchezza di features, JBuilder rappresenta ancora la soluzione migliore per chi ha bisogno di un vero RAD per il linguaggio di SUN. L'IDE si presenta nel puro stile RAD di Borland, che come si sa è il capostipite del genere. Facile da utilizzare rappresenta il miglior modo di programmare in Java sia per chi inizia sia per l'utente esperto

Directory:/JBuilder

ANT 1.6.5

IL MAKE DI JAVA

Tutti conoscono l'utility make. Il suo funzionamento è facile. Legge un file di configurazione e compila e installa un prodotto sulla base delle opzioni in esso contenute. Ant è un tool che espleta una funzione molto simile a quella di Make, ma più orientata al codice Java. Rispetto a make utilizza alcuni concetti avanzati, come la lettura delle opzioni di configurazione direttamente da un file XML. Inoltre mentre make prende parametri da una linea di comando, Ant può essere esteso direttamente da classi Java

Directory:/Ant

APACHE 2.2.3

IL WEB SERVER PIÙ USATO AL MONDO

Inutile dilungarsi in descrizioni eccessive di questo conosciutissimo strumento. Apache è il Web Server che costituisce la spina dorsale di una larghissima parte della rete Internet. In questo numero di ioProgrammo lo utilizziamo in congiunzione a PHP per mostrare l'utilizzo del nuovo Zend Framework di cui parliamo in modo estensivo. La versione che vi presentiamo è quella per Windows che non presenta difficoltà eccessive in fase di installazione e può essere utilizzata in fase di testing, oltre che in ambienti di produzione

Directory:/Apache

AXIS 1.4

IL FRAMEWORK PER I WEB SERVICES IN JAVA

Se avete provato a sviluppare qualche WS con il linguaggio di Sun, vi sarete accorti che non è propriamente un giochetto da ragazzi. Axis è il miglior strumento in circolazione per lo sviluppo, la produzione e la manutenzione di Web Services in ambito Java. Funziona benissimo con Tomcat ma può essere utilizzato anche con altri Application Server

Directory:/Axis

ECLIPSE 3.2

L'IDE TUTTOFARE

Un ambiente aperto ideato per essere un "contenitore" per lo sviluppo di applicazioni con qualunque linguaggio. Nativamente supporta Java di cui è diventato anche l'IDE di riferimento. La sua caratteristica fondamentale rimane però quella di poter essere esteso a mezzo di Plugin per poter effettuare praticamente ogni tipo di compito. Recentemente è diventato anche il prodotto di riferimento per lo sviluppo di applicazioni Flash con Adobe Flex 2.0

Directory:/Eclipse

FCKEDITOR 2.3.1

IL WORD DEL WEB

Volete inserire un editor di testi avanzato, con tanto di possibilità di formattazione, all'interno delle vostre pagine web? Quello che fa per voi è FCKeditor. Realmente il più completo editor di testi OnLine oggi disponibile. E' compatibile al 100% con Internet Explorer, ma da poco anche con gli altri Browser, vedi Firefox e Opera

Directory:/FCKeditor

FIREBIRD 2.0.0

IL DATABASE VELOCE COME UN FULMINE

Ha attraversato una serie incredibile di disavventure. Acquistato da Borland è poi tornato ad essere OpenSource. Presente sul mercato da tempo immemorabile è riuscito a sopravvivere alle sue varie vicissitudine solo grazie alle sue grandi doti tecniche. E' un database velocissimo e ultraleggero, dotato però di tutte le funzioni di un server professionale.

Assolutamente da provare!
Directory:/Firebird

IBATIS 2.1.7
L'ORM SEMPLICE
Ormai lo sappiamo: linguaggio SQL e programmazione ad oggetti non vanno d'accordo. Per superare le evidenti incongruenze di questa improbabile accoppiata sono nati gli ORM, Object Relational Mapping, tool in grado di eseguire una mappatura di elementi SQL in corrispondenti oggetti e classi. Il capostipite di questa progenie di tool è stato senza dubbio Hibernate, ma se non volete cozzare contro la complessità di questo enorme strumento allora iBatis è quello che fa per voi. E' leggero, semplice da usare e dispone di tutte le funzionalità di un ORM professionale. Usatelo se SQL comincia ad andarvi stretto
Directory:/iBatis

JDK 1.5.0 UPDATE 8
L'INDISPENSABILE PER PROGRAMMARE IN JAVA
Il JDK contiene le classi, i tool e il compilatore per poter generare programmi Java. Ovviamente senza questo strumento non è possibile in alcun modo ottenere software eseguibile. Siamo così giunti all'update 8, ricca di novità, stabile e veloce, ma soprattutto l'ennesimo passo avanti verso il JDK 1.6 ormai alle porte
Directory:/J2SE

JAMES 2.2.0
IL MAIL SERVER DI APACHE
Completamente scritto in Java è dotato di tutte le caratteristiche essenziali di un ottimo Web Server. Gestisce SMTP e POP3, filtri, blacklist ed è sufficientemente performante da poter essere utilizzato in fase di produzione. Quel che più conta è estendibile perciò facilmente controllabile e personalizzabile da chi ne fa uso
Directory:/ James

JASPER REPORTS 1.2.5
IL CREATORE DI REPORTS
Ce ne parla approfonditamente Federico Paparoni in questo stesso numero di ioProgrammo. Si tratta di uno strumento molto raffinato che consente di creare Report complessi partendo da una qualunque base di dati. E' facilmente integrabile in Java e leggendo l'articolo vi accorgerete che si tratta probabilmente dello strumento più sofisticato per la creazione di report da questo linguaggio
Directory:/Jasper

JBOSS 4.0.4
L'Application Server per le imprese
Volete sviluppare un'applicazione Web di livello Enterprise? Che faccia uso di J2EE ? Non potete prescindere da un Application Server. JBOss è il più usato dalle imprese che hanno bisogno di stabilità, sicurezza, affidabilità. Consente ovviamente di "hostare" applicazioni Java e ce ne parla ,collateralmente all'articolo su SEAM: il framework per Java Enterprise Edition, Ivan Venuti in questo stesso numero di ioProgrammo
Directory:/Jboss

JAVA SERVICE WRAPPER
PER CREARE "SERVIZI" WINDOWS
Un servizio è un programma che gira in background costantemente sulla macchina che lo ospita. Java non dispone di funzioni native che consentano ai software sviluppati con questo linguaggio di girare come demoni o servizi, così nasce questo Java Service Wrapper che si occupa appunto di "agganciare" un software sviluppato in Java al sistema di modo che possa girare in background.
Ce ne parla approfonditamente in questo stesso numero Fabrizio Fortino.
Directory:/JSW

PHALANGER 2.0
IL COMPILATORE PHP PER .NET
Si sa che .NET è una tecnologia che non dipende da un particolare linguaggio di sviluppo. Un'applicazione .NET viene compilata con una sorta di metacode per cui è possibile utilizzare un qualsiasi linguaggio ed ottenere applicazioni .NET. Phalanger implementa appunto il PHP per .NET. Facilmente integrabile in Visual Studio vi consentirà di sviluppare applicazioni PHP compilate in teconologia .NET
Directory:/Phalanger

LUCENE 2.0.0
IL SEGUGIO TROVA-TUTTO
"Ricerca delle informazioni" questa è la direzione che tutti i grandi stanno inseguendo nello sviluppo dei loro software. Il leader è ovviamente Google con il suo motore di ricerca Online e con Google Desktop, ma anche tutti gli altri produttori stanno inserendo potenti funzionalità di ricerca all'interno dei propri software, da Microsoft a IBM. Se nei nostri software vogliamo inserire questo genere di opportunità, la libreria che fa per noi è Lucene. Veloce e affidabile, in questo numero di ioProgrammo vi forniamo la versione per Java, ne esistono implementazioni anche per .NET e C++
Directory:/lucene

MYSQL 5.0.24
IL DATABASE DI INTERNET
Probabilmente il DB più diffuso nella rete. Insieme ad Apache e PHP fornisce l'infrastruttura base per una quantità enorme dei siti Web che oggi costituisce la rete così come la conosciamo. Le sue caratteristiche sono: leggerezza, velocità, stabilità. Nell'ultimo anno ha compiuto grandi passi avanti nella disponibilità di strumenti che fino a qualche tempo fa erano prerogativa solo di database più evoluti. Dispone oggi di stored procedure, viste e transazioni. Tutti elementi che insieme alle sue già innate caratteristiche lo portano ai vertici di gradimento fra i server di database
Directory:/Mysql

PHP 5.1.5
IL RE DEL WEB
Amato dagli sviluppatori OpenSource di tutto il mondo, PHP è sicuramente il linguaggio più utilizzato su Internet. Unisce una quantità industriale di costrutti base alla possibilità di essere programmato in modo procedurale e recentemente anche ad un modello ad oggetti evoluto quanto elegante. Ha una curva d'apprendimento praticamente nulla e rappresenta senza dubbio il punto di riferimento per chi vuole iniziare a programmare per il web senza troppi intoppi
Directory:/PHP

POSTGRESQL 8.1.4
IL GRATIS COMPLETO E VELOCE
Nessun server di database offre la completezza delle funzioni esposte da PostgreSQL e rimane comunque completamente Gratis. PostgreSQL è probabilmente il più estendibile fra i database esistenti. Inutile parlare della gamma praticamente completa delle sue funzioni. Il punto di forza che lo pone probabilmente al di sopra di tutti i concorrenti rimane l'alta possibilità

di personalizzazione, oltre, naturalmente, alla velocità, alla stabilità ed al costo nullo. Unica pecca, una certa complessità nella gestione. Ma sicuramente da usare in ambienti di produzione professionali che non possono accontentarsi di alcun compromesso

Directory:/PostgreSQL

RUBY 1.8.4

IL NUOVO CHE AVANZA

Per anni ha vissuto nell'ombra, utilizzato principalmente da matematici e cultori della programmazione. Oggi Ruby è diventato il linguaggio emergente, facile da usare ed elegante, vive sull'onda dell'entusiasmo per Ruby On Rails, il framework rapido per lo sviluppo Web. Nel numero di Aprile di ioProgrammo lo abbiamo utilizzato per sviluppare un plugin per Skype, ma non mancheremo di parlarne ancora in modo approfondito

Directory:/Ruby

RUBYGEMS 0.9.0

IL GESTORE DI PACCHETTI DI RUBY

Funziona un po' come apt o rpm per Linux. E' a linea di comando, prende come parametro il nome del package da installare, effettua un collegamento internet, scarica il software e lo installa. Davvero molto comodo e facile da usare. E' anche il miglior modo per installare RubyOnRails, ovviamente un package per Ruby

Directory:/Rubygems

JBOSS SEAM 1.0.1

IL FRAMEWORK PER LE APPLICAZIONI JAVA ENTERPRISE

Ce ne parla approfonditamente Ivan Venuti nel suo bell'articolo presente in questo stesso numero di ioProgrammo. Si tratta di un framework che facilita lo sviluppo di applicazioni EE in ambito Java. Nell'articolo Ivan Venuti mostra quanto questo genere di tecnica sia potente, utile e sofisticata

Directory:/Jboss-seam

SQLWEBTOOLS

AMMINISTRA SQL SERVER DAL WEB

Siete amministratori di un sistema? Volete concedere ai vostri utenti un'interfaccia comoda, rapida, veloce per amministrare i propri database SQL Server? Ecco quello che fa per voi. Completa e veloce consente di gestire gli utenti, ma anche di creare tabelle, di modificare i dati ed eseguire interrogazioni. Davvero molto utile

Directory:/ SQLWebTools

STRUTS 1.2.9

IL FRAMEWORK MVC PER JAVA

Il pattern MVC è probabilmente il più usato da tutti gli sviluppatori in ogni linguaggio. Questo framework è il leader per quanto riguarda lo sviluppo di applicazioni Java secondo il pattern MVC. Molto comodo, stabile e affidabile, rappresenta la base di partenza per applicazioni che devono essere facilmente manutenibili. Uno standard da usare se progettate applicazioni di dimensioni generose, ma anche se volete sviluppare web application professionali

Directory:/struts

APACHE TOMCAT 5.5.17

L'APPLICATION SERVER PER JAVA

Sviluppate web application in Java? Utilizzate JSP? Avete bisogno di un application server! Tomcat è il più diffuso in assoluto e anche il più semplice da utilizzare. Nato da una costola dell'Apache Project rappresenta attualmente la soluzione più comoda per quanto riguarda lo sviluppo, in molti casi può anche essere utilizzato in fase di produzione

Directory:/Tomcat

VELOCITY 1.4

IL TEMPLATE ENGINE PER JAVA

Come si fa a separare la logica di business dall'interfaccia dell'applicazione? A risolvere questo problema arriva velocity. Scrivete l'applicazione, progettate l'interfaccia e velocity mette in comunicazione i due elementi. Utile e intuitivo rappresenta la soluzione ottimale quando non si vogliono usare mostri sacri come JSF

Directory:/velocity

ZEND FRAMEWORK 0.15

UN NUOVO STILE PER PHP

Ne parliamo lungamente in questo stesso numero di ioProgrammo. E' il framework che si rivolge alla seconda generazione di programmatori PHP, coloro che vogliono partire da una base solida ed affermata e che non necessariamente devono costruire da zero le proprie classi.

Un'innovazione molto interessante per PHP. Se avrete la pazienza di leggere l'articolo, scoprirete quanto questo tipo di programmazione possa aumentare la vostra produttività

Directory:/Zend

WIX 2.0.4415

IL TOOL PER I SETUO

Nato da Microsoft e poi reso disponibile come OpenSource ecco un tool molto interessante che consente di creare script di setup molto sofisticati in modo rapido ed efficace. Non sarà sofisticato e RAD eppure si tratta del motore che fa da base per tutti i software dello stesso genere più diffusi

Directory:/Wix

ASP.NET ACTION PACK

UN MODELLO PER LA PROGRAMMAZIONE WEB

Ruby On Rails insegna, per programmare rapidamente bisogna attenersi a certe convenzioni e disporre di tools in grado di sviluppare al posto nostro seguendo le poche indicazioni che gli forniremo. Asp.NET Action Pack è un tentativo di portare questa logica in ambienti di programmazione web per .NET. Sembra anche molto ben riuscito…

Directory:/ActionPack

ASCEND.NET WINDOWS FORM CONTROL

UNA COLLEZIONE DI CONTROLLI MOLTO UTILE

Volete estendere il vostro ambiente di programmazione? Ecco una serie di controlli per le Windows Form, piuttosto ben strutturata, si và dall'uso dei pannelli decorati con gradienti fino pannelli di navigazione molto evoluti. Veramente comoda per migliorare l'aspetto estetico delle proprie applicazioni

Directory:/Ascend

C# STUDIO 2

UN AMBIENTE FREE PER PROGRAMMARE IN C#

Volete iniziare a programmare in C# ma Visual Studio vi sembra troppo complesso? Ecco C# Studio, semplice ma completo IDE per la programmazione C#. Senza dubbio un ottimo strumento anche per sviluppare applicazioni non banali. Utile se non volete ricorrere ad applicazioni più complesse

Directory:/csharpstudio

CRITTOGRAFIA QUANTISTICA

L'ULTIMA FRONTIERA NEL CAMPO DELLA SICUREZZA SI CHIAMA CRITTOGRAFIA QUANTISTICA. IL METODO È CONSIDERATO INVIOLABILE" SI TRATTA ANCHE DELL'UNICA DIFESA PER EVENTUALI ATTACCHI SFERRATI DA ELABORATORI SUPER POTENTI

REQUISITI

Conoscenze richieste
Basi di crittografia

Software
-

Impegno

Tempo di realizzazione

In questo articolo sarà la fisica dei quanti la protagonista delle nostre osservazioni. Scopriremo come si è evoluta la ricerca sugli algoritmi crittografici per elaborare difese contro eventuali attacchi lanciati da computer super potenti conosciuti come "computer quantistici". Questi utilizzando i principi della meccanica dei quanti rivoluzionano il concetto base di informazione sostituendo il bit con il qubit, ossia con un elemento in grado di rappresentare simultaneamente più stati, tale da riuscire a trattare le informazioni in modo esponenzialmente più ricco e abbattendo di fatto la complessità degli algoritmi. Una previsione questa, che se da un lato esalta l'intera comunità scientifica per i positivi riscontri nel campo della programmazione, poiché crollerebbero molti ostacoli per la realizzazione di metodi finora accantonati a causa della rilevante complessità computazionale; dall'altra allarma gli esperti di sicurezza che vedrebbero in un sol colpo demoliti i progressi nel campo della crittografia. Un computer così potente riuscirebbe dove gli attuali non hanno avuto risultati, ossia nella decifrazione di messaggi criptati ad esempio con il metodo RSA. La risposta ad un'eventuale crescita della potenza dell'hardware così repentina ha spinto gli studiosi a sviluppare un metodo di crittografia che potesse continuare a garantire la sicurezza. Così è nata la crittografia quantistica, che nello stato attuale delle cose è un metodo sorretto da una teoria ben consolidata, anche se comunque in fase di sperimentazione.

IL GATTO DI SCHRÖDINGER

Il premio nobel Erwin Schrödinger per far comprendere il principio di sovrapposizione nella meccanica quantistica amava raccontare una storiella che egli stesso aveva coniato. Si immagina di collocare un gatto in una scatola.
Il gatto si può trovare in due stati: vivo o morto. Inizialmente si colloca il felino vivo nella scatola, quindi conosciamo lo stato iniziale. Poi si infila una fiala di cianuro nella scatola. Qui si entra in una fase di non prevedibilità, poiché non si può osservare il gatto. Secondo una concezione tradizionale può essere o vivo o morto (se ahimé ha urtato e ingerito il cianuro). Secondo una logica quantistica quindi di sovrapposizione di stati il gatto e sia vivo che morto. Realizza cioè "simultaneamente" tutti e due le eventualità. La sovrapposizione permane nel periodo di non osservabilità. Quando al termine dell'esperimento si apre la scatola si conosce lo stato finale; in quel momento la sovrapposizione finisce.

FISICA DEI QUANTI IN PILLOLE

I fotoni, gli elementi che costituiscono la luce, hanno dei comportamenti davvero curiosi; talmente bizzarri che per descriverne il comportamento hanno portato alla nascita di teorie tutt'altro che intuitive. Se si osserva, come fece per la prima volta lo scienziato Thomas Young, l'interazione di un fascio di luce, ossia un insieme di fotoni, che viene fatto passare da due fenditure, si nota come i due fasci uscenti si influenzino secondo una natura ondulatoria. Il comportamento è analogo all'acqua di uno stagno in quiete segnata dal passaggio di anatre, come lo stesso scienziato fece notare. Le onde prodotte dai pennuti se si intersecano tra loro a secondo del loro orientamento possono annullarsi o aumentare di portata. Ma il comportamento di questo processo sarà stato notato da chiunque; ad esempio al mare con le scie lasciate dalle barche. È sorprendente registrare lo stesso risultato ottenuto da Young lanciando un singolo fotone ad intervalli regolari di tempo, ad esempio ogni minuto. Ci si attenderebbe un comportamento determinato dal solo percorso del fotone, invece l'insieme distinto di fotoni si comporta esattamente come il fascio di luce, ossia come se fosse presente un'interazione. Descrivere tale comportamento è stato nel corso degli anni un compito tutt'altro che semplice. Si sono comunque sviluppate due teorie. La prima conosciuta come "teoria della sovrapposizione degli stati" parte dal presupposto di conoscere lo stato del fotone quando parte e quando arriva, ma di non avere informazioni in merito al tragitto della particella luminosa e assume che possa essere passata per entrambe le fenditure. È vero che è una teoria che cozza con la logica e con tutte le "conoscenze" forniteci dalla fisica classica, ma se non altro spiega il risultato sperimentale.

Una seconda teoria ancor più futuribile è etichettata come degli universi paralleli o multiverso. Secondo questa idea il fotone viaggiando a velocità della luce si troverebbe a passare in diversi universi, che comunque interagirebbero tra loro, giustificando il risultato sperimentale. Ovviamente, non approfondiremo tali aspetti; in questo contesto ci interessa conoscere per grandi linee le teorie sviluppate al fine di comprenderne le applicazioni nel campo dell'informatica. Ci basti comunque ricordare che grazie a questa teoria si è data una risposta a molti degli enigmi per i quali la fisica classica risultava inadeguata.

UN COMPUTER DAVVERO POTENTE

Un fisico britannico, David Deutsch nel 1984 introdusse una nuova idea che sicuramente avrebbe cambiato l'approccio alla scienza in generale e alla programmazione di computer come conseguenza. Egli pensò che si poteva intendere il computer secondo una visione di fisica dei quanti e non in modo tradizionale seguendo le leggi della fisica classica. In tale teoria si supera la rigida visione binaria dei bit (appunto binary digit) e si introducono i qubit (che si pronuncia "cùbit") per i quali ogni digit segue le regole della meccanica quantistica. Così una sequenza di otto qubit esprime molte più informazioni del classico byte.
Una macchina siffatta è conosciuta come computer quantistico. Tale elaboratore consentirebbe mediante sovrapposizione di stati o se preferite con dei multiversi (universi paralleli) di valutare "simultaneamente" più possibilità, come accadeva al fotone che per sovrapposizione o trovandosi in universi paralleli produceva l'effetto di interazione contrariamente a tutte le più logiche intuizioni. Una simile risorsa apre orizzonti infiniti. Si pensi ad un possibile algoritmo come quello per il gioco degli scacchi.
È risaputo che la completa analisi delle possibili mosse prevede l'esplorazione di un albero di sottoproblemi. Poiché si devono attivare dei metodi per i quali si procede nel modo seguente: se muovo un determinato pezzo, come il cavallo, l'avversario come può rispondere? Può ad esempio effettuare dieci differenti mosse. Per ognuna delle dieci io posso effettuare almeno altrettante mosse. Insomma, l'analisi di sole poche mosse di profondità innesca una complessità esponenziale. Con un computer quantistico le dieci mosse vengono per così dire incorporate in un'unica istruzione che le valuta simultaneamente. Il risultato è l'abbattimento dei tempi di valutazione delle mosse, quindi una potenza di gioco "eccezionale". Praticamente ogni qubit andrebbe associato ad un quanto e i passaggi di stato andrebbero comandati con deboli somministrazioni di energia tali da non cambiare in modo netto lo stato dell'elemento ma capaci di innescare sovrapposizioni di stati o multiversi simili a quelle esaminate nell'esperimento. Entrando più nello specifico si potrebbero considerare come quanti gli atomi, i cui elettroni, come risaputo, sono dotati di spin. Lo spin segue una logica binaria, ovvero è la rotazione o verso est o verso ovest associabile agli stati di zero e uno. Ma trovandoci nell'ambito della meccanica quantisica non possiamo prevedere se la somministrazione di energie deboli riesca a cambiare lo spin. La realtà è comunque diversa dall'interessante teoria che detta le linee guida per la costruzione di un computer di simile potenza. La realizzazione trova molte difficoltà una delle maggiori è riuscire a mantenere una sovrapposizione quantistica durante l'intero processo di calcolo. Una seconda è l'effettivo costruzione di tali dispositivi con nano tecnologie. Cosicché, oggi si tratta ancora solo di un progetto. All'ultimo paragrafo si accenna lo stato dell'arte. È rilevante però un aspetto. Qualora si riuscisse a produrre qualche prototipo gli esperti temono ad esempio attacchi a messaggi cifrati con RSA. Si potrebbe in poco tempo demolire anche chiavi a numerosi bit. Allora è necessario correre ai ripari. Individuare metodi di crittografia sicuri capaci di parare anche gli attacchi del super computer quantistico.

UNA BANCONOTA SUPER SICURA

L'antesignano della crittografia quantistica è Stephen Wiesner che propose le sue idee sull'argomento alla fine degli anni settanta ancor prima che fosse pensato il computer quantistico. Egli ebbe un'idea che non convinse l'accademia dell'epoca e che fu quindi costretta a rimanere tale. In realtà non si occupò di crittografia quantistica, la sua congettura però costituì la base per i futuri studi a riguardo. L'allora studente della Columbia University sviluppò un progetto circa il "denaro quantistico", una particolare valuta "impossibile" da contraffare. Alla base del metodo vi è un fondamentale concetto della meccanica quantistica, il principio di indeterminazione di Heisenberg. Secondo tale principio non è possibile conoscere il presente di tutti i particoli. In altri termini se si misura ad un fissato istante la velocità non si potrà conoscere esattamente la posizione, se non un intervallo di collocazione, e viceversa se si conosce la posizione è impossibile in-

RICHARD FEYNMAN

L'idea di base circa la possibilità di sfruttare dei quanti come elemento di base per un nuovo e potente calcolatore viene da molti attribuita al famoso fisico nobel americano, Richard Feynman. Deeutch perseguì l'idea in modo più organico dimostrando che si potevano effettivamente implementare algoritmi quindi eseguire istruzioni con siffatto computer.

dividuare in modo esatto la velocità. Nel caso specifico furono osservati i fotoni che muovendosi nello spazio vibrano. È possibile stabilire l'angolo di vibrazione. Questo angolo è conosciuto anche come polarizzazione. Una normale lampadina produce fotoni con ogni possibile polarizzazione. Il piano di polarizzazione può essere verticale, orizzontale oppure tutte le posizioni intermedie. Nella teoria si semplifica collassando le posizioni a quattro possibili descritte dagli angoli rispetto al piano orizzontale. Si distinguono quindi 0°, 45°, 90° 135° che possiamo segnare con i simboli -, /, |, \. Se si collocano dei filtri di polarizzazione ad esempio uno schermo con una fessura verticale riconosce la polarizzazione | (90°). Così un filtro con fessura orizzontale riconosce i fotoni polarizzati – (0°). Una conseguenza del principio di Heisenberg è il fatto che alcune polarizzazioni riescono a passare a caso rispetto a determinati filtri. Se ad esempio usiamo un filtro verticale. Sicuramente passeranno i fotoni | e verranno bloccati i -, mentre per le polarizzazioni diagonali, alcuni passeranno mentre altri verranno bloccati in modo non prevedibile ma seguendo un presupposto stocastico secondo il quale circa il 50% passa e il restante 50% viene bloccato. Inoltre, in questo caso particolare in cui si supera il filtro la polarizzazione cambia da diagonale a verticale. Analoghi comportamenti si avranno con filtri orizzontali o diagonali. In figura 1 è riproposto un esempio di polarizzazione di 10 fotoni sottoposti a filtro verticale |.
L'idea di Wiesner affonda le sue radici proprio su questo comportamento non prevedibile tipico della meccanica dei quanti. Lo studioso concepì una speciale banconota contenente 20 piccoli dispositivi che egli battezzò "trappole di luce", capaci di catturare e bloccare un fotone. Si potevano usare quattro filtri polarizzatori così come accennati precedentemente. Così ogni banconota conteneva per ogni trappola un fotone in uno dei quattro stati: -, /, |, \. Il calcolo combinatorio ci suggerisce che le possibili differenti banconote sono un numero molto grande descritto dalla formula 4^20; il risultato è di una certa consistenza 1.099.511.627.776, più di un miliardo. Inoltre come vedremo non è richiesta neanche l'unicità della sequenza per ogni banconota. Il numero di serie della banconota che rimarrebbe nel progetto proposto si correlerebbe alla particolare sequenza di polarizzazioni, con strutture simili ad hash table, ossia semplici tabelle in cui la prima colonna conterrebbe il numero di serie e la seconda la sequenza di 20 simboli rappresentanti le polarizzazioni. Queste informazioni gelosamente custodite sarebbero di sola pertinenza della banca centrale che ha emesso la valuta. Per verificare l'autenticità di una banconota quindi basterebbe risalire attraverso la tabella alla sequenza di polarizzazioni, disporre quindi 20 filtri con uguale polarizzazione e verificare che tutti i fotoni passino. Se anche un solo fotone rimane bloccato vorrebbe dire che la banconota è contraffatta. Un ipotetico truffaldino non riuscirebbe a riprodurre le banconote poiché anche se disponesse di filtri non potrebbe usarli in modo opportuno. Infatti se ad esempio usasse un filtro verticale per testare la polarizzazione della prima cella il suo risultato non sarebbe definitivo. Esaminiamo il perché. Se il fotone passa il filtro, non è certo che sia polarizzato verticalmente |, potrebbe anche essere polarizzato in diagonale \ oppure /, di fatto si può solo escludere che sia polarizzato orizzontalmente -. Ma tale incertezza sulla polarizzazione non può portare alla falsificazione. Poiché se il falsario decide di assumere una delle tre polarizzazioni ad esempio / a solo un terzo di possibilità di avere azzeccato quella giusta. Di certo se ha sbagliato il suo errore verrà rilevato in una consistente percentuale di casi dalla verifica della banca. Per gli amanti del calcolo delle probabilità si tratta del 75% dei casi. Ora se si pensa che per ogni cella si possono commettere per 2/3 di errori moltiplicando per 20 celle è quasi impossibile che si individui la sequenza corretta.

Fig. 1: "Filtraggio di 10 fotoni polarizzati in uno dei quattro stati: -, /, |, \. I fotoni polarizzati | passano sempre mentre i polarizzati – vengono sempre bloccati. Per le polarizzazioni diagonali solo alcune volte i fotoni superano il filtraggio rimanendo per altro polarizzati verticalmente "

CRITTOGRAFIA QUANTISTICA

Curiosamente gli studi circa l'antidoto per il computer quantistico si sono evoluti molto più rapidamente e con risultati più significativi rispetto al possibile attaccante. Fu Charles Bennet un altro importante protagonista di questa storia a credere negli studi di Wiesner. Egli e Gilles Brassard colleghi dell'università canadese di Montreal riuscirono a tradurre in qualcosa di concreto le intuizioni di Wiesner su un metodo di crittografia che si basava sul denaro quantistico. Dopo anni di studi portarono a termine una ricerca che è da considerarsi un vero gioiello che non solo risponde alle esigenze avanzate dopo l'annuncio del computer quantistico ma che assume una propria autonomia nell'ambito della crittografia e che prospetta un uso di tali metodi anche su i nostri tradizionali computer, non appena alcuni limiti tecnologici saranno superati. Per comprendere il metodo, che fu battezzato BB84 come acronimo dei nomi degli studiosi e dell'anno di pubblicazione, faremo riferimento ai soliti amici Alice e Bob, che rispettivamente vogliono inviare e rice-

vere un messaggio in modo sicuro, a riparo di occhi indiscreti come quelli di Eva. Supponiamo che il messaggio sia trasmesso come sequenza binaria, a cui qualsiasi informazione si può tra l'altro ricondurre. Verranno lanciati dei fotoni in una delle quattro sopra elencate polarizzazioni: -, /, |, \. Gli 1 saranno associati alle polarizzazioni | e / e gli zeri come – e \. I fotoni inviati verranno prima filtrati con uno tra due possibili filtri conosciuti come schemi o basi. Lo schema rettilineo è l'unione dei due filtri – e |, ortogonali tra loro che indicheremo con +. Lo schema diagonale è l'insieme dei filtri \ e /, anch'essi ortogonali che verrà segnato come X. La tabella 1 mostra il valore del bit da associare al fotone da trasmettere a secondo dello schema (base).

<table>
<tr><th>Bit\base</th><th>+</th><th>X</th></tr>
<tr><td>0</td><td>-</td><td>\</td></tr>
<tr><td>1</td><td>|</td><td>/</td></tr>
</table>

Quindi se si vuole trasmettere un 1 si deve inviare il fotone | oppure / purché rispettivamente filtrati con gli schemi + e X.
La comunicazione segreta prevede che Alice invii a Bob i propri bit, ossia fotoni, usando schemi di polarizzazione scelti a caso. Una volta che Bob riceve i fotoni potrà a sua volta filtrarli. Il come filtrarli è l'idea brillante su cui si fonda il metodo BB84. Bob, infatti, sceglie anche egli a caso la sequenza di schemi da usare. Otterrà una proprio messaggio binario. Dopo, Alice comunicherà a Bob su un qualsiasi canale, anche non sicuro, lo schema usato, attenzione solo le basi e non i valori dei bit.
A questo punto Bob usando gli schemi comunicati da Alice potrà confrontarli con i propri e sapere quali sono i bit correttamente filtrati. Si può notare che anche in questo caso le leggi del calcolo delle probabilità (legge dei grandi numeri di Bernoulli) ci suggeriscono che per significative sequenze di bit la scelta corretta degli schemi di Bob è al 50%.
Così i due avranno una sequenza comune da potere usare per esempio come chiave monouso (OPT: On pad time) per un cifrario super sicuro come quello di Vernam. Una possibile intercettazione della comunicazione risulterebbe inutilizzabile per la comprensione del messaggio. Eva sarebbe costretta a usare un suo schema di filtraggio a caso, come a fatto Bob del resto ma verosimilmente diverso, potremmo usare anche sicuramente tanta è bassa la probabilità che lo schema sia identico.
Ora se anche intercettasse la comunicazione riguardo lo schema usato da Alice si troverebbe una parte di bit all'incirca il 50% come accadde a Bob per i quali i risultati sarebbero suscettibili di errori perché è stato usato una base errata. Quindi una parte di questa parte sarebbe certamente errata. In figura 2 è rappresentata la comunicazione così come descritta.

L'ATTACCO MAN IN THE MEDDLE

Se qualcuno si intrufola nella comunicazione "man in the meddle", che nel nostro esempio è woman trattandosi di Eva le cose si complicano "leggermente". Infatti, un'intrusione sulla comunicazione da parte di Eva non passerebbe indolore. Un'osservazione lascia sempre una traccia. Non si tratta di un libro giallo ma di una proprietà della fisica di elementi microscopici come i quanti. In altre parole come avevo accennato il filtro può cambiare lo stato di un fotone. Se ad esempio un fotone \ viene sottoposto a filtro | e viene riconosciuto come 1, esso subisce anche una trasforma-

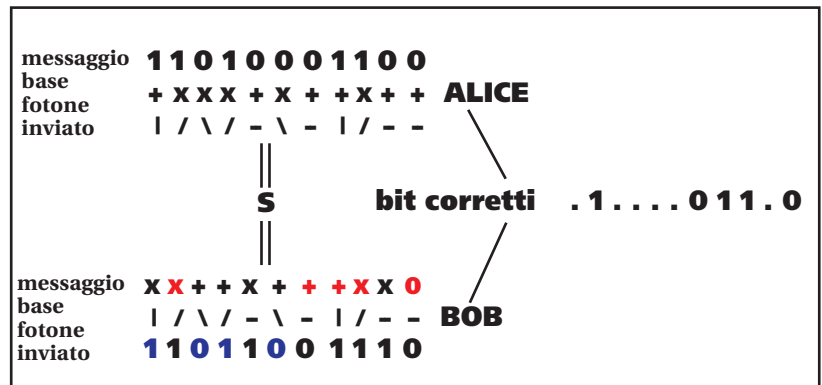



Fig. 2: "Scambio di fotoni tra Alice e Bob e scelta di filtri casuali di polarizzazione da parte di entrambi. Gli schemi di Bob sottolineati in rosso indicano quelli che sono risultati casualmente corretti. I bit sottolineati in blu, ottenuti da Bob sono corretti rispetto al messaggio di Alice, ma comunque da scartare perché ottenuti con basi differenti. I "bit corretti" sono la chiave comune"

zione di polarizzazione in verticale. Si deve quindi affrontare un nuovo problema. Se è vero che comunque Eva non sarà in grado di decifrare il messaggio ella lo inquinerà rendendo impossibile la decodifica da parte di Bob. A questo nuovo problema Bennet e Brassard diedero una soluzione molto elegante in linea con tutto il procedimento fino ad ora esaminato. Un'aliquota di bit trasmessa verrà riservata per il controllo di un'eventuale intrusione. Una percentuale tra il 5% e il 105 può essere sufficiente. Se ad esempio si trasmettono 2000 bit si possono scegliere a caso 150 bit di controllo. Alice trasmette, anche su linea non sicura la sequenza di questi 150 bit. Se Bob verifica la cor-

COSA VUOL DIRE ON TIME PAD?

Il cifrario di Vernam è un sistema crittografico basato sul quello di Vigenère. Si tratta di un sistema On time pad, ossia blocco monouso. Qui la chiave è lunga quanto il testo in chiaro. In realtà la chiave è a sua volta un testo delle stesse dimensioni del plain text. Ogni carattere descrive la sostituzione da fare per ottenere il cipher text. Ad esempio una A corrisponde a 0 una B a 1 e così via. Si veda il semplice esempio per capire come funziona.

Plain text: DIFESA
Chiave: PANICO
Cipher text: SISMUO

Il cifrario di Vernam è l'unico sistema crittografico la cui sicurezza è comprovata da una dimostrazione matematica.

rettezza di questi bit, con un'altissima probabilità si potrà stabilire che la comunicazione si è svolta a riparo da occhi o orecchi indiscreti. Quindi dalla chiave si scorporeranno i 150 bit di controllo per renderla totalmente sicura. Se i 150 bit non collimano significa che c'è stata una variazione dovuta ad un'intrusione. In tal caso bisognerà ritrasmettere l'intera chiave, magari utilizzando un diverso canale. Del resto se Eva riesce a intercettare la trasmissione circa la comunicazione dei bit di controllo questi non saranno sufficienti per la ricostruzione dell'intera chiave scambiata. Si tratta di un altro caso in cui il calcolo delle probabilità stabilisce solo a livello teorico eventuali possibilità per le quali il sistema non funziona ma che sono talmente poche, esprimono cioè probabilità così piccole, da lasciar considerare il metodo super sicuro. In definitiva il metodo può essere sintetizzato come segue:

1. Alice polarizza con una sequenza di basi casuali una sequenza di fotoni e li invia a Bob.

2. Bob ricevuti i fotoni li interpreta filtrandoli anch'egli con una sequenza casuale di filtri.

3. Alice comunica a Bob quale schema avrebbe dovuto usare per il filtraggio.

4. Bob sulla base della informazione precedente individua una serie di bit (quelli che corrispondono tra lo schema inviato da Alice e quello da egli usato) che formano una chiave comune ai due interlocutori da poter usare ad esempio per metodi OTP. La chiave così ottenuta andrà ridotta di alcuni bit come specificato al passo successivo.

5. Alice invia un campione casuale di bit di controllo a Bob che ne verifica la correttezza, in tal caso si eliminano i bit di controllo e si ottiene la chiave definitiva. Se la verifica non risulta corretta si ripete il passo 1, magari usando un altro canale di comunicazione.

FOTONI ENTANGLED

Per consentire tra due utenti lo scambio di una chiave crittografica in modo sicuro, Artur Ekert nel 1991 propose un secondo protocollo basato sul principio fisico dell'entanglement quantistico. L'algoritmo usa coppie di fotoni appunto entangled, che abbiano cioè particolari caratteristiche di correlazione, ad esempio sullo spin degli elettroni. Il metodo prevede una terza parte, una sorgente che emette con regolarità, ad esempio ogni millesimo secondo, coppie di fotoni entangled che si propaghino in direzioni opposte, uno verso Alice (mittente) e l'altro verso il Bob (destinatario). In sintesi Alice e Bob dovranno verificare le caratteristiche di fotoni che hanno ricevuto, cioè utilizzando ognuno uno schema di filtraggio, che successivamente andrà comunicato al partner. Il resto del metodo e per molti versi simile a BB84.

LO STATO DELL'ARTE

Per entrambi i progetti: il computer quantistico e la crittografia quantistica si stanno registrando un notevole fermento di interessi. Circa la realizzazione dell'hardware non vi sono molte notizie anche perché secondo alcuni ci sarebbe un interesse militare che sposterebbe la studio in ambiti riservati. Sono riconosciuti gli attenzioni da parte di DARPA (Defence Advanced Research Projects Agency) l'agenzia della difesa per i progetti avanzati. Secondo alcune teorie già esisterebbero alcuni prototipi. La cosa certa è che ancora non è mai stato presentato alcun computer con tali caratteristiche. Ha fatto comunque scalpore la dichiarazione di una società canadese la D-Wave Systems che prevede per il 2008 l'immissione sul mercato del primo computer quantistico. Per la realizzazione del primo prototipo avrebbe ricevuto cospicui finanziamenti. Un'altra linea di pensiero vede la realizzazione ancora lontana. Per il pro. Serge Haroche dell'università di Parigi VI è ingiustificata l'euforia circa l'imminente realizzazione del super computer. Per la crittografia quantistica la situazione è sicuramente in maggiore evoluzione. Già nel 1988 Bennet con l'aiuto del ricercatore John Smolin riuscì a stabilire la prima trasmissione quantistica. Certo si trattava di due computer posti ad una distanza di 30 centimetri. Successivamente, ulteriori sforzi di ricerca hanno portato a risultati più significativi. Utilizzando la fibra ottica, canale di trasmissione ideale per la bassissima ("nulla") dispersione, a Ginevra è stata stabilita una trasmissione per ben 23 chilometri. Risultati apprezzabili sono stati ottenuti all'università di Los Alamos nel Nuovo Messico, qui il canale usato è stato quello satellitare. I primi due prototipi commerciali sono appannaggio delle societò "MagiQ Technologies" (New York) e "id Quantique" (Ginevra). Ma più di recente l'Unione Europea ha finanziato il progetto SECOQC (sviluppo di una rete globale per comunicazioni sicure basate sulla cifratura quantistica), iniziato il 1 Aprile 2004 e da alcuni indicato come il progetto "anti-echelon" (un modo per contrastare lo spionaggio USA nei confronti di cittadini e aziende europee). Il budget del progetto è di tutto rispetto: 11,4 Milioni di euro in 4 anni. Partecipano 41 partner in 12 paesi europei, e per l'Italia vi sono l'Università di Pavia, il CNR, la Scuola Normale Superiore di Pisa ed il Politecnico di Milano.

CONCLUSIONI

Ciò che è stato osservato è relativo a ricerche di frontiera che potrebbero maturare in qualcosa di concreto e commerciabile nel giro di pochi anni Il computer quantistico è un concetto più astratto per il quale dobbiamo aspettare qualche altro anno. Nel frattempo molte implementazioni sono già utilizzabili, fatene buon uso

Fabio Grimaldi